DOCUMENT RESUME

ED 029 679 52 LI 001 535

By-Cunningham, Jay L.; And Others

A Study of the Organization and Search of Bibliographic Holdings Records in On-Line Computer Systems: Phase I. Final Report.

California Univ., Berkeley. Inst. of Library Research.
Spons Agency-Office of Education (DHEW), Washington, D.C. Bureau of Research.
Bureau No-BR-7-1083
Pub Date Mar 69
Grant-OEG-1-7-071083-5068
Note-307p.
EDRS Price MF-$1.25 HC-$15.45
Descriptors-*Automation, Bibliographic Citations, *Catalogs, Computer Programs, Computer Storage Devices, Costs, *Information Processing, *Information Retrieval, *Information Storage, Information Systems, *Libraries, Library Technical Processes, Search Strategies, Systems Development

This report presents the results of the initial phase of the File Organization Project, a study which focuses upon the on-line maintenance and search of the library's catalog holdings record. The focus of the project is to develop a facility for research and experimentation with the many issues of on-line file organizations and search. The first year has been primarily devoted to defining issues to be studied, developing the facility for experiment, and carrying out initial research on the issues. Achievements involved: (1) obtaining equipment; (2) programming and testing an initial software system, and then expanding it to supply access to the central processor from two different mechanical terminals at two remote locations; (3) planning for acquisition and incorporation of an existing machine file as well as bibliographic records which require original conversions; (4) developing software for data base preparation and for file handling and access; and (5) initiating analyses on issues such as optimum length of search keys. Appended are six reports which cover specific aspects of the project and an article entitled "The Organization, Maintenance and Search of Machine Files," reprinted from "The Annual Review of Information Science and Technology," volume 3. (JB)

ED029679

LI 001535

FINAL REPORT
Project No. 7-1083
Grant No. OEG-1-7-071083-5068

A STUDY OF THE ORGANIZATION AND SEARCH OF BIBLIOGRAPHIC HOLDINGS RECORDS IN ON-LINE COMPUTER SYSTEMS: PHASE I

By

Jay L. Cunningham
William D. Schieber
and
Ralph M. Shoffner

Institute of Library Research
University of California
Berkeley, California 94720

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY.

March 1969

The research reported herein was performed pursuant to a grant with the Office of Education, U.S. Department of Health, Education, and Welfare. Contractors undertaking such projects under Government sponsorship are encouraged to express freely their professional judgment in the conduct of the project. Points of view or opinions stated do not, therefore, necessarily represent official Office of Education position or policy.

LI 001535

U.S. DEPARTMENT OF
HEALTH, EDUCATION, AND WELFARE

Office of Education
Bureau of Research

TABLE OF CONTENTS

Page

List of Figures . . . . iii-vi

Acknowledgments vii

I. INTRODUCTION AND SUMMARY 1

- A. The Research Problem 1
- B. Research Method 3
- C. Significant Findings and Achievements 4
- D. Future Directions 5

II. FACILITY ESTABLISHMENT 7

- A. General 7
- B. Equipment 12
- C. Computer Programs 14
- D. Data Base Development 20
- E. File Structure 21

III. THE BIBLIOGRAPHIC RECORD 35

- A. General 35
- B. Record Content 37
- C. Record Form 38
- D. The Representation of Typographical Characters 48
- E. Logical Similarity of Bibliographic Records . 60

IV. DATA BASE DEVELOPMENT 75

- A. General 75
- B. Strategies of Conversion 77
- C. Translation of Existing Machine Files 85
- D. Data Base Production Procedure 94
- E. Issues of Cost and Quality 107

References 115

TABLE OF CONTENTS, (Cont.)

Page

APPENDIX I: AN ALGORITHM FOR NOISY MATCHES IN CATALOG SEARCHING, By James L. Dolby 117

APPENDIX II: USER'S GUIDE TO THE TERMINAL MONITOR SYSTEM (TMS), By William D. Schieber 137

APPENDIX III: A DESCRIPTION OF LYRIC, A LANGUAGE FOR REMOTE INSTRUCTION BY COMPUTER, By Steven S. Silver 145

APPENDIX IV: ILR PROCESSING RECORD SPECIFICATION, By Jay L. Cunningham 163

APPENDIX V: SUMMARY OF RECORD FORMATS FOR DATA BASES TO BE CONVERTED TO ILR PROCESSING RECORD FORMAT

1. Santa Cruz Record Format 193
2. ILR Input Record Format 207
3. Experimental On-line Mathematics Citation Data Base 263

APPENDIX VI: SAMPLE SIZE DETERMINATION FOR DATA CONVERSION QUALITY CONTROL, By Jorge Rodriguez 271

APPENDIX VII: THE ORGANIZATION, MAINTENANCE AND SEARCH OF MACHINE FILES, By Ralph M. Shoffner (Published in the Annual Review of Information Science and Technology, v. 3, edited by Carlos A. Cuadra. Chicago, Encyclopaedia Britannica, Inc., 1968. pp. 137-167) 277

LIST OF FIGURES

Figure — Title — Page

SECTION II: FACILITY ESTABLISHMENT

1. Schematic Diagram of Project Facility 8
2. File Generation Process 17
3. Blocking Strategy 24
4. Block a Non-Keyed File. 25
5. Uniqueness of Author Identification 30
6. Distribution of Number of Fields of Length N. 31
7. Schematic of Multi-level File Structure Linkage . . . 33

SECTION III: THE BIBLIOGRAPHIC RECORD

8. Functional System Components Related to Different Record Formats. 36
9. MARC II Elements Deferred in File Organization Project Data Base 39
10. File Organization Project Data Elements Not Defined in MARC II. 40
11. Coding Sheet - Monographs 46
12. Keying Blocks of Text 52
13. The Tentative Harvard List of Diacritics. 56
14. Alphabetical Index of Diacritic Codes 57-58
15. Proposed Single Keying Codes Compatible with Transliteration Schemes for Modern Cyrillic 59
16. A Spelling Equivalent Abbreviation Algorithm for Personal Names (Dolby Version 1 - Variable Length). . 65
17. Equivalence Class Computation (Manual). 66
18. Equivalence Class Computation (Computer). 68
19. Abbreviation Algorithm for Personal Names (Version 2 - Fixed Length). 69

LIST OF FIGURES (Cont.)

Figure | Title | Page

SECTION IV: DATA BASE DEVELOPMENT

20. Distribution of Duplicate Titles as a Function of Publication Date:
    A. Titles in English Language 86
    B. Titles in Languages Other Than English That Use a Roman Alphabet 87
    C. Titles in Languages That Use a Non-Roman Alphabet 87
21. Conventional Conversion Compared to Automatic Format Translation and Computer-Assisted Editing. . 89
22. Flow Chart of Personal Author Field Algorithm . . . 91
23. Flow Chart of Title Field Algorithm 92-93
24. Summary Chart of Data Base Production 95
25. On-line Search for Duplicates 97
26. Verification of Match 98
27. Data Preparation and Transcription. 100
28. Computer Edit, Correction Cycle, and File Update. . 101
29. Diagnostic Printout, Part 1 - Logical Field Listing 105
30. Diagnostic Printout, Part 2 - Card Image Listing. . 106
31. Schematic of Quality Control Subsystem. 108
32. Relation of Initial Keying Cost to Accuracy 113
33. Acceptability in Terms of Accuracy and Cost for Three Price Quotations for Keying 114

APPENDIX IV: ILR PROCESSING RECORD SPECIFICATION

1. Indicator for Main Entry - Personal Name. 166
2. Storage Record Organization 166

LIST OF FIGURES (Cont.)

Figure	Title	Page
3.	Schematic of ILR Storage Record, INFOCAL Version 1 .	168
4.	ILR Processing Record - Segment 1, Leader.	169
5.	ILR Processing Record - Segment 2, Record Directory.	172
6.	ILR Processing Record - Segment 3, Fixed Length Data Elements.	173
7.	Variable Field Tags and Data Elements.	177
8.	Values for Indicator 1 in Applicable Fields.	180
9.	Sub-Field Delimiter Codes.	184
10.	Proposed Variable Field Header	188

APPENDIX V-1: SANTA CRUZ RECORD FORMAT

1.	Sample Catalog Record in Original Santa Cruz Format.	195

APPENDIX V-2: ILR INPUT RECORD FORMAT

1.	Storage Record Components & Organization	208
2.	Structural Patterns in MARC Record Data Definition .	211
3.	Example of Input Format Mapping Into Processing Format	213
4.	Example of Tab Card Decklet - ILR Input Format . . .	217
5.	ILR Input Record Format - I-Fields: Data Elements & Codes.	219
6.	ILR Input Record Format - A-Fields: Data Elements & Codes.	224
7.	ILR Input Record Format - B-Fields: Data Elements & Codes.	226
8.	Input Code Values Table for Type of Main Entry . . .	232
9.	Input Code Values Table for Type of Added Entries (Series Traced Same)	233
10.	Input Code Values Table for Type of Added Entries (Subject Added Entries).	234

LIST OF FIGURES (Cont.)

Figure	Title	Page
11.	Input Code Values Table for Type of Added Entries (Other Added Entries)	236
12.	Input Code Values Table for Type of Added Entries (Series Traced Differently)	238
13.	Presence of Fields in an Input Record	239
14.	INFOCAL Default Initializations	241
15.	Default Settings for Indicator 1.	243
16.	List of Tag Numbers Which are Currently Repeatable.	244
17.	Revised Field Coding: A-Fields & B-Fields.	246
18.	Table of Valid Symbols.	250

APPENDIX V-3: EXPERIMENTAL ON-LINE MATHEMATICS CITATION DATA BASE

1.	Cards Punched for One Paper Published in Vol. 66 of the Communications in Pure and Applied Mathematics.	268

ACKNOWLEDGMENTS

This report comprises the results of the first year of effort under a grant, OEG-1-7-071083-5068, from the Bureau of Research, Office of Education, U.S. Department of Health, Education and Welfare. The content and conclusions presented in the report pertain to the period July 1, 1967 - June 30, 1968. The University of California also provided contributory support. M.E. Maron, Associate Director of the Institute, acted as Principal Investigator and Ralph M. Shoffner as Project Director.

For constructive criticism concerning goals and methods, and for otherwise inaccessible information, we are especially grateful to the members of the project's Consultant Advisory Panel. The members of this Panel were: Mrs. Henriette D. Avram, Library of Congress; Richard Dougherty, University of Colorado Libraries; Anthony Hall, UCLA University Research Library; Foster Palmer, Harvard University Libraries; Charles Payne, University of Chicago Libraries; Charles Stevens, Project INTREX, Massachusetts Institute of Technology; and Allen Veaner, Stanford University Libraries. Valuable consultant support in the development of an algorithm for searching through "noise" was also received from James L. Dolby, of The R & D Consultants Company, Los Altos, California.

Our campus Computer Center has been most helpful in planning and obtaining the system needed to support the project. Our initial monographic data base was supplied through the cooperation of Donald Clark, the University Librarian of the Santa Cruz campus, without whose help we could not have had access to such a large file. The data base of the citation index in mathematics was supplied by Mary Tompkins of the Institute's Los Angeles staff.

We would also like to give credit to the individual contributions made by staff members and research assistants of the Institute, in particular Harriet Zais, who performed statistical studies of the data base and who supervised the pilot testing of the input coding system; and Thomas Hargrove, who designed the scheme for conversion of foreign language materials and much of the procedure for the data base production. Technical studies were produced by Jorge Rodriguez and Naresh Kripalani. Programming support was rendered by Arjun Aiyer, Regina Frey, John Reinke and Steve Silver. Input editors who got the original conversion task started were Mrs. Ann Giglioli, Diane Kristell, Lucy Liang, Elizabeth Poole, Janet Redd, Roberta Roberts, and Douglas Romney.

Finally, we would like to thank the Institute office staff who were instrumental in the physical preparation of this report: Patricia Barkeley, Linda Child, Kitty Colburn, Betty Geer, Marion Gordon, Judith Sutliff, and Connie Torii.

J.L.C.
W.D.S.
R.M.S.

I. INTRODUCTION AND SUMMARY

A. THE RESEARCH PROBLEM

1. Prologue. This is a period of major intellectual and technological change in libraries. Among the many aspects of this change, the application of computers to library operations is of recognized importance. Much effort is currently being directed toward such use of computers. For example, there are a growing number of files of bibliographic records which are being provided in computer processable form by both public and private organizations. Most forecasts of the library of the future include on-line computer use as a key feature of the library's operation. Attractive as such forecasts may appear, there are numerous unresolved and often ill-defined problems which must be solved before on-line systems can be effectively applied to the acquisition, maintenance, and retrieval of recorded information. This report presents the results of the initial phase of a research study, the File Organization Project, which focuses upon the on-line maintenance and search of the library's central apparatus of bibliographic control, the catalog holdings record.

Machine files are being sought and created as a feasible solution to the problem of growth and complexity in the catalog, as a response to the need for new and expanded services, for their speed and convenience in access, and to replace human labor in generating products such as book-form catalogs and bibliographies. What are the dimensions of the task of acquiring, maintaining and using the very large data base? Why is it desirable or even necessary that library files be accessible on-line? What techniques can be recommended as efficient, economical and acceptable for organizing and searching large files in operating contexts?

2. The Economic Implications. This question about techniques can be expanded into other questions. What methods should be used to encode record content in the various parts of the system -- input processing, storage, output? What should the file structure be, that is, how will the records be mapped into the physical storage and how should they be related to each other? What should be the form and capability of the search language? What post-retrieval analysis and processing capability should be provided to the system user?

To select appropriate techniques in answer to these questions one must face all of the issues of evaluation which are common to information retrieval research, that is, by some method the cost of the system must be considered in relation to its performance. Leaving aside the problems of establishing the system cost, there remain myriad problems to be resolved in establishing the performance of the system. For example, how does one measure the effect of a system upon its user? Can recall and relevance or similar such measures be applied? Should one consider in the performance of the system the "non-users" of the system, or the questions that for various reasons are not or cannot be asked of it? There are only certain aspects of performance that can be measured. The problem then is to establish a relationship among them in order to provide a suitable measure; for example, how can system capacity, actual use, average response times, response time variations, etc., be combined in a performance measure? These questions can be in turn expanded into still other questions. However, though they are but a sample of the questions which do not at present have operational answers, they are among the most important and they form the context for our study.

On-line time-shared computer systems present many desirable features for library use: parallel operations in multiple locations, response times to complex queries measured in minutes or seconds, reduction in the manual labor required for the maintenance of catalogs, etc. At the same time, man-effort is required to design and implement these systems. Further, the operating costs of an on-line system are high when compared to those of a computer system organized for conventional batch processing. That is, a batch processing operation can always be organized in such a way as to be less expensive to operate than an on-line system, on the condition that no cost be assigned to the processing delay incurred by the batch system.

An estimate of the operating cost of an on-line system for a library can be obtained by considering the major cost factors of the system: the terminals, the mass-storage for the bibliographic records, and the computer time used. The costs of terminals and the interfacing equipment necessary to attach them to the central processor vary widely depending upon type of gear, total number of terminal units, data transfer rate, distance from the processor installation, and the like. For purposes of rough estimation, however, a reasonable system operating cost lies in the region of $2000 per terminal per year. Due to the size of the bibliographic record and the character of present auxiliary mass storage devices, the cost of storing the data equivalent to that of one catalog card ranges from $0.10 per record per year, to $0.60 or more. The cost of the central processor allocated to the terminal network during the time it is in use is similarly highly variable. At present, these costs range from $2 per terminal hour to $10 and beyond.

A library with 100,000 master bibliographic records to be accessed in an on-line mode might need 20 terminals or more with an average utilization of 1200 terminal-hours per terminal per year. Thus, the following minimum costs for such a configuration can conceivably be incurred:

Terminals	20 @ $2000/yr.	$ 40,000
Mass Storage Capacity	100,000 recs. @ $0.30 ea.	30,000
CPU Time Charges	20 term. x $1200/ term. yr. @$2.00/ term. hr.	48,000
	TOTAL	$118,000/yr.

Due to the fixed sizes of the various central processors and auxiliary storage units now available, it is most likely that the actual charges would be higher still. In any event, it is clear that such costs are significant. There will be real monetary impact on libraries resulting from improvements in the processing capacity of the computer system employed to maintain and search the bibliographic files.

Should libraries undertake the development of such systems? The question has in a sense become moot in that there are already so many on-line system development efforts. Rather than attempt to answer this very broad question, the File Organization Project is directed toward an analysis and understanding of specific issues of organization and search if an on-line system is used. Such understanding should then contribute to the establishment of the most appropriate blend of computer services for a given library.

B. RESEARCH METHOD

The focus of our efforts in the File Organization Project is to develop a facility for research and testing, one within which experimentation with the many issues of on-line file organization and search can be performed. In this first phase of the project we have been concerned primarily with the design of the facility and the implementation of its initial components. In addition, to ensure the development of an adequate facility we must establish central examples of the crucial issues which are to be investigated. Thus, the research method has had the following points: to define precisely the issues which will be studied, to develop the facility within which such issues can be studied, and to carry out the initial research on these issues.

Our approach has been to organize both the tasks necessary to the establishment of a usable facility for testing, and the tasks directly assigned to attacking the experimental issues in such a way as to keep the tasks manageable and within reasonable boundaries, hopefully without sacrificing the quality of the ultimate results of the project. The elements of the facility have been decomposed into separate units for ease and speed of accomplishment, particularly where programming is involved. The research issues have been compartmentalized and broken down into isolable units so that their respective dimensions and sub-problems are delineated for the staff member assigned as problem solver. An insight into the effectiveness of this approach can be gained by a simple enumeration of the significant findings and achievements of the first year.

C. SIGNIFICANT FINDINGS AND ACHIEVEMENTS

1. Facility for Experiment.

a. General. Equipment was obtained and an initial software system was programmed and operated on a test basis to demonstrate capacity to search (by author only) a file of 75,000 catalog records on-line. The system was expanded to supply access to the central processor from two different mechanical terminals at two remote locations on the Berkeley campus. Mass storage equipment with capacity in excess of 200 million characters was procured. The necessary adaptation units for interfacing with remote terminals were installed at the campus computer center to support the activities of the project. Plans were laid for extension and upgrading of the system to handle the complex requirements of visual (CRT) terminals, now on order.

b. Programs. File structure design was specified and programmed for an initial configuration of multi-level index files linked to the master data base, using manufacturer supported software. Search strategy development was initiated and preliminary programming of Boolean search programs commenced.

c. Data base. Plans were completed for acquisition and incorporation of an existing machine file and for the incorporation of bibliographic records requiring original conversions. We expect that within a year our data base will contain in excess of 120,000 records.

A logical record format for input conversion of catalog records not in the existing machine file was designed and a program written to accomplish conversion to a unitary storage format.

A storage format was designed having special features to handle special characters and diacritical marks in any language likely to appear in textual data, yet retaining convertibility to and from the Library of Congress' MARC II communications format.

2. Analytic Issues. Analyses were initiated on a number of the most pressing issues, in particular the optimum length of search keys, e.g., author name. This analysis will be used in establishing index files and in guiding users in the minimization of keyboarding time and effort. This analysis represents the first step in deciding how to allocate storage for both the master file and the index files, based on the various critical factors such as frequency of access, equipment access time, and storage costs.

User aids were under development, in particular an algorithm which will serve as the core of a sub-system designed to process and assist the terminal user in overcoming the effects of spelling errors. Analysis of these aids will follow their development.

D. FUTURE DIRECTIONS

The general purpose of setting up the facility is to perform analyses and experiments in on-line file organization and search of bibliographic records. Experimental facilities are often needed to collect data and test hypotheses. In the present instance such a facility is especially important in order to record information about factors such as terminal user behavior and the frequency of recurrence of requests, as well as to test our analytical models against actual system operations.

This facility will enable us to experiment with users who have real needs for information and thereby to relate our research to its operational implications. To accomplish this, we will place terminals in areas where they can be used by library patrons. Both the placement and the period of use of these terminals will be determined by the experiments planned. It is anticipated that these terminals will remain in place for at least three months to allow time for patrons to learn how to use them.

A number of conditions must be fulfilled before such "live" testing can be undertaken. The three most important are the realization of the basic search and retrieval system organized around the visual (CRT) displays and the basic internal record formats, the internal (to ILR) experimental use of this system, and the modification and improvement of the system prior to its experimental use by library patrons. We expect that at least a year and a half will be required before making the system available for experiments with users outside the Institute of

Library Research. This is because of the need for time to make the system reliable and its use easy enough for it to be accepted by library users. Thus, the second phase of the study will consist primarily of a continuation of the "setting up" activities concerned with the facility and the associated support components, such as software for data base preparation and for file handling and access. In addition, a considerable amount of development and analysis is needed with respect to the following:

To expand the internal processing format to accommodate augmented bibliographic records for non-monograph materials.

To implement a data compression method to convert back and forth from internal processing format to mass storage format.

To expand the formats to encompass Cyrillic records.

To implement a reasonably efficient program system as a foundation for the ready development of user interface programs.

To implement user assistance routines such as those providing proper name equivalence classes.

To implement storage allocation algorithms based upon quantitative analysis of the file parameters.

Such a list can be indefinitely expanded. However, all of the above have been given at least initial effort during the current phase and we hope to accomplish work on all of them during the coming phase.

II. FACILITY ESTABLISHMENT

A. GENERAL

A facility for experimenting with the on-line organization and search of bibliographic records, requires equipment, programs, and data bases. Work has been performed during the first phase to plan this facility and to begin development of it. In addition, some analysis using the facility has been initiated. An overview of the facility and its major components is schematically depicted in Fig. 1.

1. Equipment. We plan to implement the facility on two closely related sets of equipment: the IBM 360 model 40 computer system with mass storage and cathode ray tube display terminals at Berkeley; and the IBM 360 model 75 with similar peripheral equipment at Los Angeles. At present we are utilizing two mechanical terminals in Berkeley. The Los Angeles system has IBM 2260 displays.

2. Programs. Initial programming has been performed on many aspects of the system. We will continue this programming and implement it on both computer systems. The following are examples of programming that has been done or that is planned:

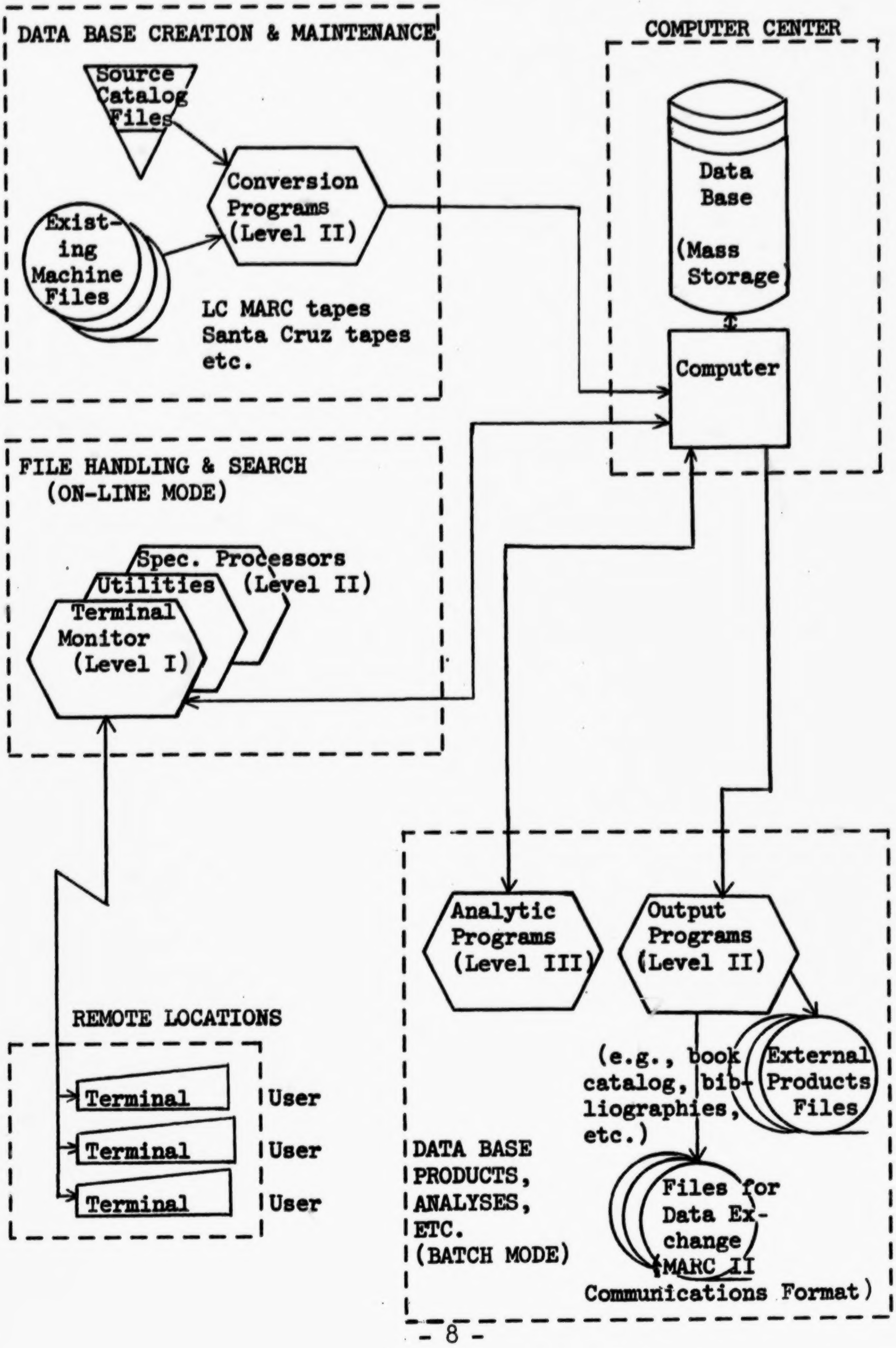
a. Monitor system. At present, monitor systems to provide terminal operations are operative at both Berkeley and Los Angeles. The systems are not designed to the same conventions, however. In order to implement common programs to operate within these systems, we plan to modify the Berkeley monitor system to bring it into conformance with the more advanced Los Angeles monitor.

b. Data base programs. Programs are under development both to provide the data base (both records and file structure) and to search it. These programs are now being used for monographic files and will be used for journal article records. Initially the file access programming has been restricted to the development of a basic search and retrieval system utilizing temporary formats for monograph and journal article records. Following this, we will extend the system to utilize our internal storage format. This format has been fully defined for monographic records. However, it will be extended to include other types of records in the file structure.

Programming of file handling operations will continue throughout the study. As soon as the basic search system is in full operation, however, we will make it available for internal (to ILR) use in the data base development effort. We will augment the existing computer programs, dividing the effort between the internal file handling operations and the user interface procedures. To guide part of this augmentation,

FIGURE 1:

SCHEMATIC DIAGRAM OF PROJECT FACILITY


DATA BASE CREATION & MAINTENANCE
Source Catalog Files
Conversion Programs (Level II)
Existing Machine Files
LC MARC tapes
Santa Cruz tapes
etc.
COMPUTER CENTER
Data Base
(Mass Storage)
Computer
FILE HANDLING & SEARCH (ON-LINE MODE)
Spec. Processors
Utilities
(Level II)
Terminal Monitor (Level I)
REMOTE LOCATIONS
Terminal
User
Terminal
User
Terminal
User
Analytic Programs (Level III)
Output Programs (Level II)
(e.g., book catalog, bibliographies, etc.)
External Products Files
Files for Data Exchange (MARC II Communications Format)
DATA BASE PRODUCTS, ANALYSES, ETC. (BATCH MODE)

we are now developing a quantitative analysis of the effect of file structure on retrieval operations. We will then develop programs to set up the alternative file structures which our analysis indicates to be of interest.

As soon as our basic search and retrieval system is fully available, we will monitor the performance of the system users. In this monitoring process, we will keep a precise machine-readable record of the two-way interchange between the user and the system. We are now planning how to keep and identify this "history" record so that useful analyses can be performed.

As an example, one such investigation to be carried out is that of determining common errors which are made in keying bibliographic data. In addition to using the terminal system to obtain data on error, we will study our off-line data base input operation to analyze the nature of errors which are made in conversion keying. Following this we will develop automatic correction routines and study their behavior. While the correction routines would have to be combinatorial in nature if they are intended to be exhaustive, it is our hypothesis that certain types of errors occur with considerably higher frequency than do others, and therefore useful heuristic routines can be developed even though they may be partial in their effect.

Such analyses are expected to lead to the development of routines which can assist the non-expert user in obtaining the materials he desires. Therefore, we anticipate programming these routines and incorporating them as part of the search system available to the general user.

We anticipate that large amounts of programming will be required to provide the user interaction programs. To support this programming we are developing the LYRIC processor (at Los Angeles).* This processor will facilitate the rapid development of the user interface programs. PILOT, a similar processor, is being programmed in PL/1 at UC, San Francisco. Although we anticipate that LYRIC will be preferable, we will experiment with both processors.

c. Analytic programs. To support our research, many special-purpose programs will be required. The analysis of error mentioned above will require programs of this type. Others will be concerned with analysis to investigate the issues discussed in the Introduction, such as record encoding, file structure, etc. These will continue to be defined and developed throughout the study.

3. Data Base Progress and Plans. The data base development, initiated during the current phase, will be continued. The monograph records will comprise the largest portion of the data base. As an initial data base for the project, we have

---

*See Appendix III.

utilized 75,000 catalog records which were available in machine form from the Library of the University of California Santa Cruz campus, and 40,000 records of the Mathematics Citation Index available at Los Angeles. These records have been utilized in their original format. However, we have been developing record formats to be used on a continuing basis for the facility. The record formats have been developed to be compatible with the Library of Congress' MARC II format in order to provide the greatest likelihood that the records and programs will be of use to others. The results of this work are presented in Section III.

We will continue to develop our data base in order to obtain a file of a size that can support meaningful experimental work with regular library users. Our objective is to obtain an on-line file of at least 500,000 bibliographic records. During the second phase of the project we hope to complete the establishment of a file containing a minimum of 1/4 million records, from three sources: by obtaining MARC II records, by original input, and by converting the Santa Cruz records.

We have developed procedures for original input of monographic records. We are considering a production plan which utilizes both on-line terminals for the search of the existing data base and off-line terminals for the large scale keying of original input. We have defined a statistical quality control procedure for maintaining the accuracy of this input process. During Phase II we plan to test, modify and re-test both the extraction and the original input procedures.

Because there are a number of existing bibliographic files in different machine formats, we have been investigating the problem of utilizing such files through the method of routines for automatic conversion of such files to a common format. Our first task has been to develop computer routines for converting the Santa Cruz format (a relatively simple format) to the MARC II format (a complicated format). Through of dubious practical value unless the file to be converted is of considerable size (we believe it to be in excess of 100,000 records), the problem is of considerable research interest since its basis is the recognition by computer of the components of a bibliographic entry.

Though data base development is not one of the central concerns of our study of on-line file organization and search, there are significant problems involved in such development, and the inescapable requirement to develop a data base has immersed us deeply in these problems. Because many of these problems are of general interest but are not dependent upon the rest of our work, we have dealt with our solutions to them separately in Section IV.

4. Analysis of File Structure. An analysis of file structure was initiated recently. Our objective is to create

an analytical model of the retrieval system, which will delineate the relationships between access time, the cost of direct-access storage and the strategy for allocating space. Initially this study will be limited to trade-off analyses between a few variables. We will then test the model by organizing the file in the manner indicated and check the outcome of the predictions. This model will, for example, provide information on when it is more economical to divide a logical index file into two or more physical index files, in mass storage. To support this work, we have observed the relationship between the length of the beginning portion of an identification key (e.g., author name) and the degree of uniqueness this identification provides in the retrieval of records from the Santa Cruz file. For certain purposes, the key might ultimately be composed of portions of several fields of the record (see Chicago search code*).

In order to make it possible to experiment with alternative file designs, we must have flexible programs which are capable of maintenance and retrieval without re-programming even though the file structure is modified. To accomplish this, we will continue the attempt to develop a structure which is capable of containing a wide range of content, and from which any record or sequence of records can be retrieved by using a common retrieval routine. The programs have been designed to be parameter-driven, so that changes in file structure may be easily accommodated.

5. Other Analyses. We have barely begun to identify and study the central issues associated with on-line organization and search. Among these, the grouping of records or parts of records by similarity or other relations pertaining either to subject content or to user need, has received growing emphasis in the Project.

We have programmed the first of a set of routines for the equivalence class coding for author names, title words and other substantives (reported in detail in Section III).

Associative indexing by statistically-generated means, as a technique for providing improved retrieval, can be effective for material which is indexed in depth, i.e., with a large average number of assigned index terms per document. We plan to investigate the utility of such associative techniques on monographic records, which by present cataloging practice have a much lower average number of assigned index terms per item - i.e., subject headings per document. As part of this study, the subject headings in the file will be analyzed for characteristics such as distribution by date of publication, co-occurrence, and the number of headings per document.

---

*Payne, Charles T. "Tagging Codes." Chicago, University of Chicago Library, Feb. 1967. (unpublished report) various pagings.

Our work on formats tailored to the various system functions, i.e., input, processing, etc., leads naturally to concern for data compression techniques. Our concern here is to provide efficient representations of the records when they are in the mass storage device. We plan to initiate a task to analyze the applicability of known data compression techniques in terms of their encoding and decoding effort together with the resulting amount of space required to represent bibliographic records.

B. EQUIPMENT

1. Characteristics. In selecting a system for our experiments with bibliographic data, we must trade off the following six desirable characteristics:

a. The maximum amount of large-capacity, random-access storage capacity that can be attached to a central processor.

b. The least difficulty in attaching remote terminals, of both the mechanical and cathode ray tube (CRT) variety.

c. Machine logic capable of handling the maximum number of individual characters in order to facilitate handling of the wide range of typographic characters encountered in bibliographic data.

d. That to the utmost extent possible, system software especially in the realm of physical I/O, be provided by the manufacturer of the machine.

e. That this full system be available on a time-shared basis during as much as possible of the regular working day.

f. That it be the most common system in use for bibliographic data, in order to share software.

In choosing between the two systems available to us at the Berkeley campus computing center (an IBM 360, model 40 and a CDC 6400), the IBM 360 provided the better compromise among this set of desired characteristics.

2. Main Frame. The central processing unit (CPU) of our 360/40 has a 128K byte memory. Of this, the Operating System takes approximately 30K. The remaining core storage is divided into two partitions. In one of these (approximately 20K in size), utility programs are run (such as tape copy operations). The remaining 80K bytes are available for other applications. It is in this larger second partition that we operate when on-line.

3. Mass Storage. The concept of the on-line system necessarily involves the use of random-access mass storage. Three types are generally available for 360 attachment: drum, disk, and magnetic strip (also called data cell). Drum storage was rejected because of its limited capacity. Magnetic strip storage

was considered, but rejected for use during Phase I of the project because of persisting hardware failures. (These problems, we understand, are currently being corrected.)

Given our requirement for very large random-access storage, the IBM 2314 disk storage facility appeared particularly satisfactory since it is a supported unit in terms of IBM software and has high operational reliability. This device provides a maximum storage capacity of 233 million characters. However, the device's effective capacity is approximately 150 million characters since some space is required by the system in order to store the operating system programs and control data. The access time (i.e., the average time required to position the access device to the desired record) for the device is 100 milliseconds. The data transfer rate (i.e., rate at which data is moved between the device and core storage) is 312,000 characters per second.

4. Terminal Equipment. In our experiments, we have planned that inquiries made to the file will be processed over remote terminal equipment. This means that processing programs must operate in a tele-processing environment where several terminals are busy at the same time. Since the transfer rate of characters over phone lines to a terminal is relatively slow, the computer central processing unit (CPU) while servicing one terminal is waiting virtually all of the time. It is this waiting state or unused CPU time that makes time-sharing with multiple terminals an attractive alternative, since with the attachment of a few additional terminals the CPU is utilized more fully, with little noticeable increase in delay time to individual terminals.

Currently, we have two mechanical terminals attached to the 360: a Teletype model 35 ASR and an IBM 2740 communications terminal. Data transfer rate to each terminal is approximately 12 and 15 characters per second, respectively. Each has a private line connection over voice-grade telephone lines. The teletype can read and punch paper tape, a feature useful in situations where data can be prepared before going on-line, in order to obtain the maximum input rate when on-line.

Mechanical terminals, as we have indicated, produce hard copy output which in some instances may be desirable. However, in our early work it has become evident that the slowness of a mechanical terminal as it types out, character by character, the results of an inquiry, does not make it particularly attractive for most library retrieval applications. Therefore, we are now planning to connect CRT terminals for inquiry and display of search results. As a shared task among several ILR projects we have completed an exhaustive inquiry into the characteristics of currently available CRT's. A major drawback at this time is that, with the exception of extremely expensive terminals, the maximum number of displayable characters is under 100, a number not nearly sufficient to meet our capacity requirements for the display of bibliographic data. Even so, the high volume of data output associated with bibliographic search makes it desirable to incorporate CRT's as soon as possible, in order to facilitate testing on a basis superior to that achievable with the mechanical devices.

C. COMPUTER PROGRAMS

Three levels of programs are required for the facility we are developing. On the first and most general level is the monitor system which provides for remote terminal operation. The second level programs carry out the various operations of file generation, organization, maintenance, retrieval and display. The third level contains programs which analyze the performance of the system, to guide its development. The second and third level programs do not necessarily operate in an on-line mode.

1. Terminal Monitor System.

a. Introduction. During the course of the year we have developed an experimental Terminal Monitor System (TMS) which is designed to facilitate both communication between remote terminals connected via phone line to the Computer Center's IBM 360/40, and data transfer between the 360/40 and the attached disk storage facility. All files are maintained on ILR's private disk facilities and are not accessible by other 360 users.

TMS has a time-sharing design which allows multiple terminals to operate seemingly at the same time. It allows the user to carry on a limited dialog with the computer, and will wait for the user to enter his response before continuing processing. TMS operates as a user subsystem of the manufacturer-supported Operating System/360 (OS) in a partition of core storage of approximately 80,000 bytes.

TMS performs five general functions:

(1) Text entry: the establishment of new files which can later be processed.

(2) File search: retrieval and display of records from within an existing machine file.

(3) Text editing: addition, replacement, and deletion of character strings, individual records, and blocks of records within an existing file.

(4) Compilation of source programs: conversion to executable instructions from source language (FORTRAN, PL/1, and 360 Assembler) entered in the manner of text.

(5) Interface to special user-written routines: capability for terminal user to load and execute special-purpose programs.

The logical structure of TMS is built around a set of two supervisory routines which maintain control over a series of processing programs (called processors). It is the processors which perform the file handling functions listed above.

b. Supervisory routines. One of the two supervisory routines (TXIO) has the function of coordinating input/output operations to the terminals attached to the system. It does such things as directing message transmission to the proper terminal, and the analysis of which user has sent a message when one is received. It also performs error recovery and recording (when a transmission error has occurred), character code translation on both incoming and outgoing messages (since each different type of terminal has its own character code), and message attribute and length analysis.

The other supervisory routine (BASE) is the overseer of both TXIO and the set of processors. Its function is to bring a processor into core storage when needed by a terminal user. After loading a processor, the BASE routine delegates control of the terminal to the processor, which then communicates directly with TXIO to accomplish terminal input/output. If more than one terminal requests the same processor for simultaneous use, the BASE routine does not load another copy; both terminals use the same copy. The arrangement which allows sharing of a processor in this way is called "re-entrant coding". The amount of core storage in use at any one time is thus a function of the number of terminals on-line, the space required by each terminal's processor, and the degree to which terminals are sharing processors.

Processing programs are of two sub-types: 1) utility processors, which perform general functions and are used by many users and 2) specialized processors written by terminal users for their own file handling applications.

c. Utility processors. Currently there are seven utility processors. The specific instructions for existing processor use, showing the formats of both TMS messages and terminal user's responses, appear in the Terminal User's Manual (See Appendix II). The first three processors are used for general file handling operations. The last four are used in the development of new, user-written processors. The processors are:

(1) Text processor. Enables the terminal user to create a new file. Each record in this file is 80 characters in length, of which 76 characters may contain user data and 4 characters the key by which a record can be retrieved.

(2) Search processor. Used to display one or more records from an existing keyed file. It has the ability to search on the full key or on a portion of a key.

(3) Edit processor. Used to edit existing files. It is possible to perform replacement, addition, and deletion functions on character strings within a record. In addition, one can also add or delete entire records.

(4) Assembler language processor. Here a source language file created as text is given over to the 360 Assembler to generate object (machine language) instructions. This, and the following PL/1 processor, are used to generate new programs which may become part of the processor library.

(5) PL/1 Language processor. Used to generate object code from PL/1 source language stored as text.

(6) A link processor. Has as its input object code (generated from either the PL/1 compiler or the Assembler) and generates a new (or replacement) processor.

(7) LYRIC. A processor for user interaction routines, which is not yet fully implemented (See Appendix III).

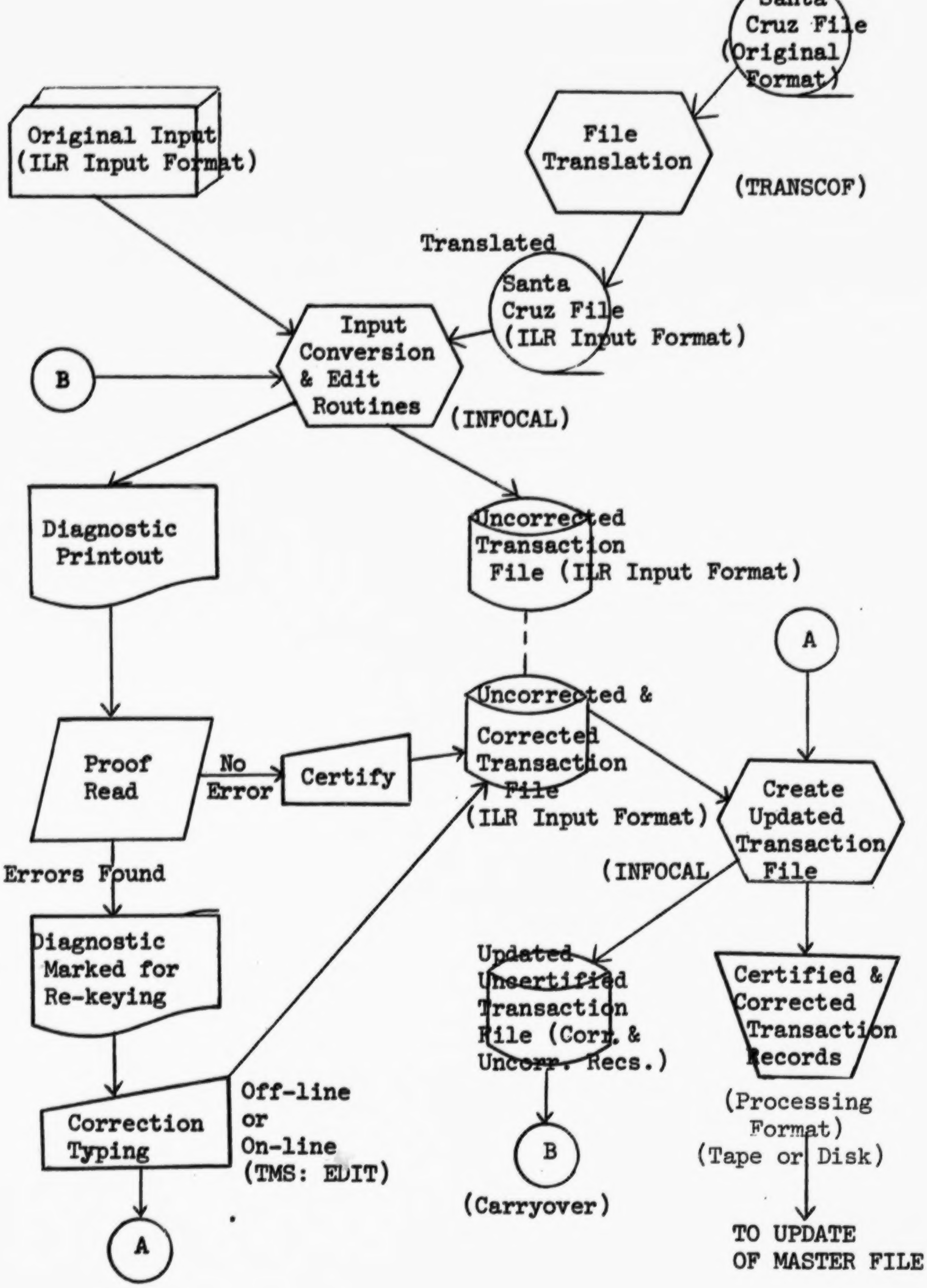
d. Specialized processors. Currently there are approximately ten specialized processors. The most important of these, the file handling programs, are discussed below.

All processors exist as load modules, and are maintained as individual members of a partitioned file. Each processor is planned to be re-entrant, so that multiple use does not require separate copies of a given processor. A terminal user writing a specialized processor must adhere to several conventions: He must observe standard calling conventions, and must provide error recovery which returns control directly to the calling program (the BASE routine). Terminal I/O is accomplished through calls to special interface routines which are link-edited into the user's routine during execution of the LINK processor.

2. File Handling Programs. Two classes of programs have been written which have as their focus the development and use of the project's data base. The first class is concerned with the generation and maintenance of the master file and its associated index files. These programs are batch-oriented and are not operated as part of the Terminal Monitor System. The second category consists of programs which are terminal based (i.e., processors) and which are concerned primarily with retrieval from the existing bibliographic files.

a. File generation programs. Fig. 2 shows the file generation process. The names in parentheses identify computer routines required. These file generation programs provide for both original input and extraction from existing machine files.

FIGURE 2: FILE GENERATION PROCESS


Santa Cruz File (Original Format)
File Translation
(TRANSCOF)
Original Input (ILR Input Format)
Translated Santa Cruz File (ILR Input Format)
Input Conversion & Edit Routines
(INFOCAL)
B
Diagnostic Printout
Uncorrected Transaction File (ILR Input Format)
A
Proof Read
No Error
Certify
Uncorrected & Corrected Transaction File (ILR Input Format)
Create Updated Transaction File
(INFOCAL)
Errors Found
Diagnostic Marked for Re-keying
Updated Uncertified Transaction File (Corr. & Uncorr. Recs.)
Certified & Corrected Transaction Records
(Processing Format) (Tape or Disk)
Correction Typing
Off-line or On-line (TMS: EDIT)
B
(Carryover)
A
TO UPDATE OF MASTER FILE

The most important of the programs written for this purpose is the one called INFOCAL, which converts data presented in input format to internal processing format. INFOCAL is written in PL/1. Unfortunately, it is so large it is very slow. For continuing operation, it appears that an assembly language routine will be required. This program is designed to do several things before a record is actually placed in the master file. First, every new record entering the system may, as an option, be printed out on an edit listing, so that it can be proofread for errors in the text. Second, the program does a certain amount of syntactical checking of field codes and other control information, and prints diagnostic messages on the edit listing. At the same time the program also places a copy of each new record in a transaction file written on disk storage as a keyed file.

Corrections are written on the edit listing by the proofreader and changes to the file may be keyboarded either on-line via remote terminal, using EDIT, the edit processor available under the TMS, or off-line for processing in a batch-oriented update routine. If a record has no further corrections, it may be certified either off-line or from the terminal as ready to be transferred from the transaction file to the master file. On the next successive execution of the INFOCAL routine, records marked as certified will be written out and are ready to be added to the master file. Routines to accomplish this maintenance cycle are now in the design stage.

Records carried over from the existing transaction file, which are not yet certified, must go along with new records into the new (updated) transaction file.

The file translation program, TRANSCOF, converts the existing Santa Cruz data base from its original record format to the ILR input format (that is, the record structure accepted by INFOCAL). This routine, still under development, is being written in assembly language as a result of our operating experience with PL/1. We planned to produce our input format so that if there are changes to the internal format, only INFOCAL will need to be changed. Additional discussion of TRANSCOF is provided in Section IV.

b. Specialized retrieval processors. In order to begin experimentation with terminal-based search algorithms, we placed 75,000 records from the Santa Cruz file on the 2314 disk facility. These records are still in the original format, not yet having been converted to ILR internal format. To retrieve records, we have written a primitive processor (SCAT) which allows a terminal user to type an author's name over the terminal, and to receive a list of documents indexed under that name. Although we have not generated them, other search keys could be included readily. However, we did not include them in this initial test because Boolean combinations of the terms would not be possible with SCAT.

For the user, the search technique is quite simple (and, of course, limited). The contents of the file can be displayed by typing an author's name. Retrieval will be made of all records for which the left-most characters of the key match those characters input over the terminal by the requestor. For example, if the requestor typed the string "GARN" when asked to place his request, the program would retrieve all records whose key begins with these characters. Currently, three authors match: "Garn, Stanley M.", "Garner, Wendel R.", and "Garnett, Arthur Campbell". If he had typed "GARNETT", he would, of course, match only on "GARNETT, ARTHUR CAMPBELL".

The master Santa Cruz file contains full bibliographic entries. When printed out on the terminal each citation occupies from five to twelve print lines. It becomes evident very quickly that, given a mechanical terminal, some means must immediately be constructed to enable the user to limit the amount of material which is typed out before him. Display of a full record of average size (400-500 characters) consumes about 1-1/2 minutes. Ten of these would require 15 minutes. For the future, it would thus appear that mechanical terminals will not be suitable for on-line retrieval of bibliographic data.

Because of such time delays, we have initially provided two mechanisms for limiting output. One of these is to request the user to indicate how many entries he will accept in response to his search request. The other is to allow him to display index entries only. An index entry contains only one line of data which gives the full key (full author name) and the address of the master file entry (its sequence number). Upon completion of the index file search, the user may then request full citations by record number.

Although the SCAT processor is quite limited, it has provided us with immediate experience which has been useful in determining the types of retrieval capability to be incorporated in future search programs. Further, we now have had experience with multi-level file structures of the type we will be using throughout the study.

3. Analytic Programs.

a. General purpose. The third level contains the programs concerned with the analysis of the system - its content, structure and operation. These "third level" programs are not a component part of the system providing file maintenance and search; rather, they operate independently to obtain information to guide the project.

Although no programming on this level has yet been accomplished, we anticipate that some of these programs will be used quite generally. For example, one kind of general program will be for extracting information from a history file containing the records of past system use. This history file will be maintained as a basic part of our terminal handling program. In this way we will accumulate a history file which can be used to make the same sequence of inquiries against file structures which have been developed using different organizing strategies. In these reruns the results of the searches will not go out over the terminals. Instead, we will record the time for the storage access portion of the query processing in order to obtain comparable timings for the different structures. We will use this approach to study factors such as prediction of relative frequency of access to the records in the file, as a basis for reorganizing the files in order to minimize average access time for the individual query.

b. Special purpose. To support our research, many special purpose programs will be required. Examples of these programs are those to analyze the frequency of co-occurrence of index terms in the data base; the time pattern of requests for file information; and the effects of a file compression technique on storage space and encoding - decoding time. Programming of such routines will continue throughout the project.

D. DATA BASE DEVELOPMENT

1. Monograph Data Bases. To begin the study, a brief survey was performed to determine the availability of library catalog files which might be used. While there are numerous such files, those of UC Santa Cruz library, Stanford University Undergraduate Library and LC MARC I* seemed most complete and accessible. Arrangements were made to obtain all three files. However, the Stanford file has not yet been obtained.

Before our work with the MARC I records had begun, the Library of Congress announced the MARC II format. Therefore, we decided to organize the major work of the project around this new format and in the interim until it is available to concentrate on the use of the Santa Cruz file. A brief description of the Santa Cruz format is provided in Appendix V. Section III of our report provides the results of our work which concerns the design of these records. Section IV provides a description of our resolution of the problems associated with the development of a large data base.

2. Other Data Bases. In addition to the task of acquiring existing files of monographic materials, a further goal of the project is to study the problems of integrating and searching files of records for other materials, e.g., journal articles,

---

*The Institute at Los Angeles was a participant in the MARC Pilot Project experiment.

music, etc. The availability of bibliographic machine files on the level of the journal article is placing increasing pressure upon the library community to provide computer-based access to these articles. Thus the issue of organization and search of such files is as pressing as that for monographic material. At the same time, it does not appear to require a uniquely different approach to its solution. Therefore, we are including journal article machine files in the study.

Although these records necessarily vary in content from that of monograph catalog records, compatible record formats may be developed. This is a goal to which the Library of Congress MARC Project has addressed itself in the development of a common machine format for bibliographic and other library materials. That is, the MARC II format is designed as a data exchange medium, with a structure intended to have "wide applicability to all types of bibliographic data" and to be "hospitable to all kinds of bibliographic information." For "any given type of item, e.g., serials, maps, music, etc., the components of the (MARC II) format may have specific meanings and unique characteristics."*

Whether it will be possible to so integrate the different records remains to be seen. However, we are working on the integration of a citation index file which is a non-monograph data base. Work has been carried on at UCLA to extend the Citation Index in Mathematics, which was initiated at Princeton. The nature of the UCLA data base and its preparation is described in the Appendix V. At this time, we have no significant results to report.

E. FILE STRUCTURE

1. Inverted File Structure. In the first phase of this study we began a detailed examination of file structure. Early work in this area indicates that the way in which files of bibliographic information are structured profoundly influences both the time required to gain access to a record in the file and amount of storage space consumed by the files. In a large file it is evident that one cannot conveniently make a sequential search of the entire file in order to satisfy a unit request for information. As a result, files usually have been structured in what is called an "inverted structure". In this structure all logical records are held in a "master file" with auxiliary files (called index files) created which allow quasi-random access into the master file.

---

*Avram, Henriette D., John F. Knapp, and Lucia J. Rather. The MARC II Format: A Communications Format for Bibliographic Data. Washington, D.C. Information Systems Office, Library of Congress, 1968. p. 10.

It is possible to define a master file structure from which records could be retrieved by some given portion of it, called a key: for example, author. However, since in most computer systems a record has only one access point, its address, we must define additional files which provide multiple access points into the master record. These additional files, the index files, contain records which point to and allow us to locate and access the information contained in master records.

In our system the full bibliographic record comprises the master record. The content and format of each master record is described in detail in Section III. The full set of master records will be called the master file. The content of an index record consists of the index term (the access point), and pointers, one to each master record which has been indexed under that particular term. One record in an author index file might, for example, be:

SMITH, JOHN W.	A_1	A_2	A_3	A_4

In this record, SMITH, JOHN W., is the access point to the index records and A_1, A_2, A_3 and A_4 represent pointers to four different master records which have been indexed under this access point. Because index terms have varying numbers of documents indexed under them, the length of the logical index record is variable.

For our initial operations, this is the type of structure we have been using. However, this approach is not adequate to our purposes in developing a system for experiment. During the course of the study we will want to maintain maximum flexibility in the operation of the system. We will want to split logical records, create new index files, partition them, relocate records and files, and make new groupings of records. Such manipulations will allow us to explore the relationships which exist between access time and storage cost. For this reason we are developing a more general system employing the basic file structure described below. As a part of this we are writing a set of utility retrieval routines which operate independently of file content.

2. Random-Access Device Characteristics. In our experiments we are using a large-capacity disk facility to store the bibliographic data files used. Since the way the file is structured is, in large part, determined by the methods with which data can be physically stored on a disk, it is necessary to discuss briefly the ways in which information can be stored on and retrieved from a disk device.

a. Storage modes. Data is recorded on a disk in recording surfaces called tracks, all of which are of equal capacity in the 2314 facility. Movement of data written onto or read from a track is done in units called "block" (also referred to as physical records*). Normally, the size of a block cannot exceed maximum track capacity. We have now experimented with two general types of file: 1) keyed files and 2) non-keyed files.

A keyed file is one in which each logical record can be retrieved by supplying some unique portion of it called a key. In general, each record may have only one key which must be identified at the time the file is created. Non-keyed files may be retrieved only by address. This address may be the actual position of the record on the device (that is, the track number where the record resides), or it may be a relative address (that is, its position relative to the beginning of the file). Actual addresses are rarely used, since this technique makes a file unmovable. Relative addressing allows the file to be moved to any position within the device.

Whether keyed or non-keyed, the very large size of our files and the relatively high cost of direct-access storage, make it desirable that the data be placed on the disk in as compact a form as possible.

Disk space is used most effectively when logical records are blocked to maximum track capacity, and all blocks are of fixed (i.e., equal) length. This is because fixed-length blocks involve less overhead both in terms of access time and storage space than do variable-length records. Blocking to maximum track capacity also involves less device overhead since each block must have a given amount of system control information stored with it.

To illustrate: Track capacity of a hypothetical device is 3000 characters; and each block must have 50 characters of system control information stored with it. If we allow only one block per track, we must allocate 50 of the 3000 characters for control information. This leaves us 2950 characters for the block of data. If we allow two blocks per track, the system requires 100 characters with 2900 remaining for the two blocks (1450 characters in each). Fig. 3 shows figures for differing lengths.

---

*As distinct from logical records; i.e., logical units of data.

FIGURE 3: BLOCKING STRATEGY

Track capacity = 3000 characters
System overhead = 50 chars./block

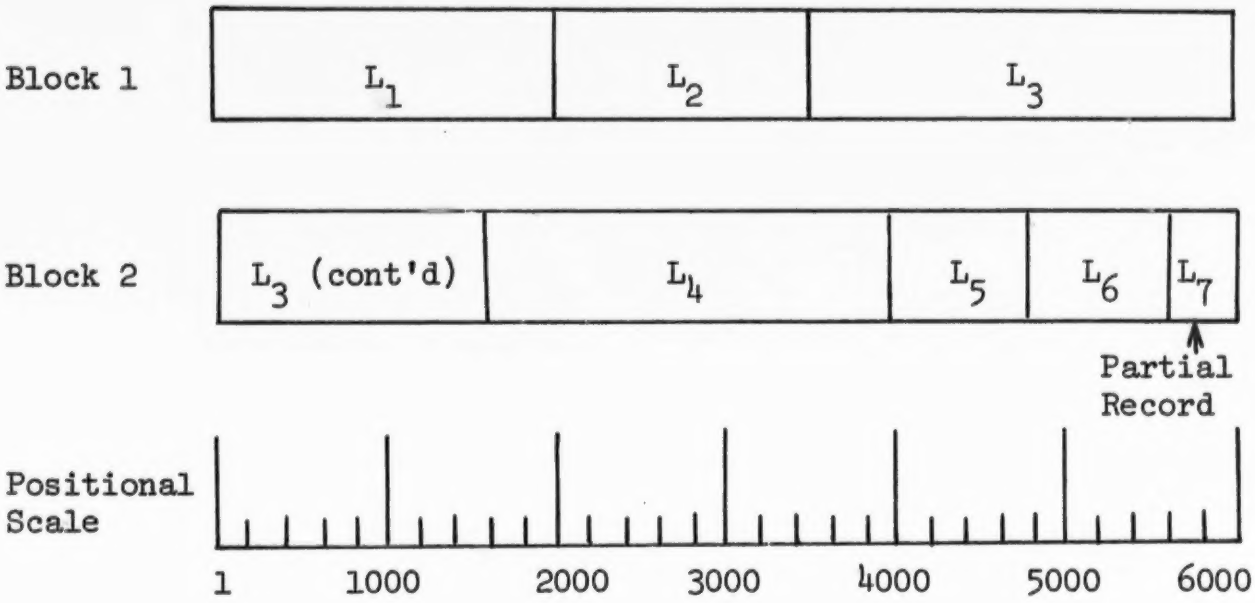
Number of Blocks per Track	Total Allocation for System Overhead	Total Space Remaining for Data	Size of Each Block
1	50 char	2950 char	2950 char
2	100	2900	1450
3	150	2850	950
4	200	2800	700

To summarize: All blocks, whether in a keyed or non-keyed file, should be both fixed-length and blocked to maximum track capacity.

Since most bibliographic records are variable in length, and we wish to use fixed-length blocks, we are adopting a technique which translates variable-length logical records into fixed-length blocks of maximum track capacity. A different strategy is required for keyed files than for non-keyed files.

b. Non-keyed files. This structure involves sequential placement of the variable-length logical records into a block sized buffer in main storage; when that buffer is full, the entire block is written to disk. In order to retrieve a logical record we need to know: 1) in which block it begins, 2) its relative position within the block, and 3) its length. In the example in Fig. 4, a total of six complete logical records (denoted by L_n) and part of a seventh record have been mapped into two physical blocks.

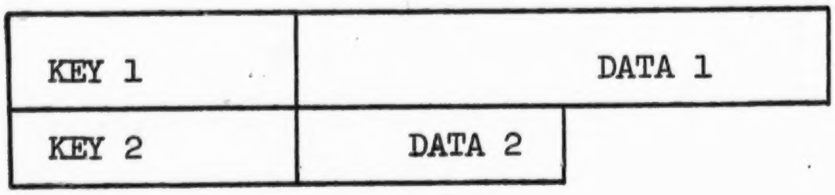
FIGURE 4: BLOCKING A NON-KEYED FILE



The first logical record (L_1) is of length 2000 and can be retrieved by first reading block 1, then locating position 1 within the block. Similarly, the second logical record (L_2) can be addressed by Block 1, Position 2000. It has a length value of 1500 characters. Note that it is necessary to break some records across track boundaries, so that one logical record may actually be contained in two (or possibly more) blocks. Therefore, when a logical record is broken across track boundaries, a signal must be inserted to indicate whether or not the record continues into the next block.

Thus, any variable length logical record can be retrieved by supplying its block (or track) number, its position within the block, and its length, which together constitute the address of the record.

c. Keyed files. To map variable-length keyed files into fixed-length keyed format*, we separate each logical record into two components. The first component, a fixed-length, keyed record, points to the variable length data portion. For example, the two records:



---

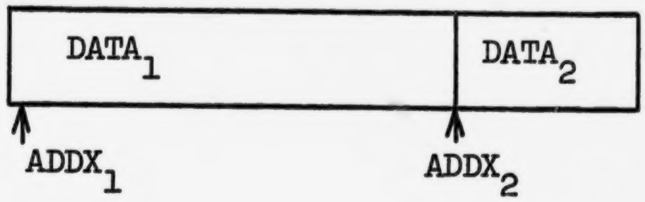
*The operating system data management facilities currently require the index entry in a keyed file to be fixed-length, specifiable by the user.

would each be split into two records. The first set is the keyed records:

KEY 1	ADDX$_1$
KEY 2	ADDX$_2$

ADDX is a pointer field that contains the address of the variable length data portion.

The second set of records contains the data portions:



These are placed in a variable-length, non-keyed file and treated as discussed above.

3. Multilevel File Structure and File Linkage. Central to an understanding of a general retrieval routine are the concepts of record segmentation, file segmentation, and linkage. An index file may, for example, be composed of two distinct files which are linked together. The mechanism which links two segments together is called a pointer. In the simple inverted file structure, the pointers in the index record connect to addresses of master records. In the same way, two distinct physical files may be linked to form a single index file. Similarly, a master record itself might be divided into two or more parts, one part containing those elements of the logical record which are used frequently (e.g., main entry, title statement, imprint and call number) and the other portion(s) of it which are required with lesser frequency (bibliographic notes, etc.) Thus it is possible in each case to have a record which has both content and a pointer to another record, or portion of a record. We shall call the individual records which are linked together (by pointers) linkage segments. The linkage segment is conceptually independent of its logical content. This file structure is similar to that of a linked list file. In the conventional version of the latter file structure, the master records themselves are usually not segmented.

Why should one wish to subdivide a master record? One can answer this question both at the level of our requirement to develop a system for experimentation, and on the level of the design of an operational on-line bibliographical retrieval system. Taking the latter first, it is most likely that the variable length records will be segmented into fixed block sizes in order to simplify the problems of input/output and storage allocation. If one places the most frequently used parts of the record in the first segment then following segments will not need to be obtained as often as they would if the allocation to segment were independent of use. Thus, the capacity and response of the system would be improved. This improvement would be even more dramatic if two random-access storage facilities are used, with the high frequency elements allocated to the one which is fast and expensive, while the lower frequency elements are allocated to the other slower and less expensive unit. Such storage of segments on the basis of access time of the device we call "horizontal segmentation".

One can also perform "vertical segmentation", that is, within the file, the records themselves may be arrayed by frequency and divided into groups containing the most-used and least frequently-used records. Here again one could place the most frequently used records on the fast device and the least-used on the slow device.

The answer for our facility derives from the comments just made. These remarks indicate a conceptual solution to the problems of file organization. However, it is a solution which raises a host of questions about the relation of segment sizes, element frequencies, access time and cost, and storage costs. These questions can only be answered by analysis and experiment. Therefore, we wish to set up our computer routines such that varieties of segmentation can be accommodated in order to be able to carry out relevant experiments.

For our initial experiments we have chosen a simple file structure. It consists of:

a. A master file consisting of records stored without keys.

b. Index files for author, title and subject. Each index file consists of two sub-files.

(1) a keyed file (called the "access file") in which the access point is the key and the data field contains a single pointer to the second sub-file.

(2) the "address file", a non-keyed file containing a variable number of pointers to the master records which have been indexed under the term represented in the access file.

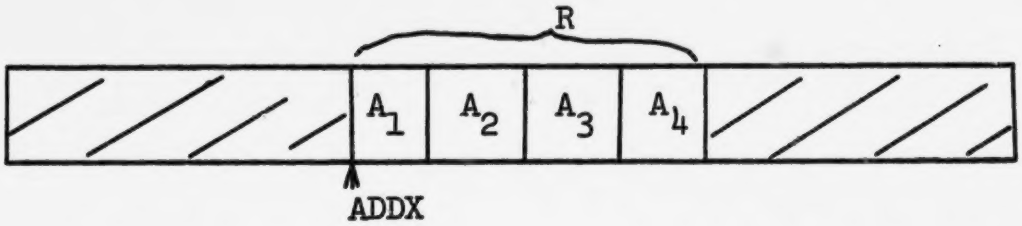
To illustrate: Four master records have been indexed under the term CARTOGRAPHY.

The entry in the access file for this term would be:

CARTOGRAPHY	ADDX

where the term CARTOGRAPHY occupies the key portion of the record, and ADDX is the data portion containing the pointer to the corresponding record in the address file. Logical records in the access file will be of fixed length, while those in the address file will be variable length.

The address file entry for this term will have four fields containing the addresses of the master records. The format will be:



where the shaded part indicates other data contained in the track, R is the record, and A_1, A_2, A_3, A_4 are pointers which contain the addresses of the four master records.

4. Uniqueness of Identification as a Function of Key Length. A significant issue in the study is the number of characters which are required to uniquely identify a record held in the system. This is important for several reasons. First, the more characters which must be input through a terminal, the longer the encoding process becomes and the less capacity the terminal will have for handling requests. Second, the more characters which must be transmitted by the terminal, the greater the chance for keying error becomes. At best, these errors will be automatically corrected,* but invoking correction routines will reduce the capacity of the system even further. At worst, these errors will initially cause improper retrieval and thus a partial failure of the system. Finally, the expected key length for uniqueness could be used as a basis for index record segmentation. The objective of the system, therefore, should be to provide search based on enough characters in the request key to provide a reasonable amount of uniqueness of the resulting search output, but with as few characters as possible in order to reduce the problems of keyboard time and keyboard error.

We have analyzed the amount of uniqueness of author names as a function of the number of characters of input provided. The program used distributes the number of catalog records per unique search tag of given length for author keys. The results, taken from a sample of 20,000 author names, are presented in Fig. 5. We plan to run this analysis on title and subject key fields as well, in order to determine the proper length for those search keys. At present we are implementing our search routines so that a search key of any length can be used by the searcher. Consequently, our search strategy is independent of search key length, which enables us to experiment with keys of various lengths.

---

*For example, by use of equivalence class coding techniques discussed in Section III.E.

The segmentation of an index file may be performed on the basis of key length. In that case, the distribution of the length of the index terms, or keys, determines the effect of segmentation. In preparation for the analysis of the effects of segmenting the index files, we counted the lengths of the call number field, the author field, and the subject heading fields in the Santa Cruz machine file. Fig. 6 contains the distributions of the field lengths for those elements.*

5. Linkage Control in Retrieval. The retrieval routine needs to provide access to logical records from two types of files: keyed and non-keyed; in addition it must be able to distinguish between data which is content and pointer data. If a record contains pointer data the routine must know whether or not to go on retrieving subsequent records in the linkage sequence. In some cases presentation of information contained in the pointer field will be enough to satisfy the control routine.

The pointer field establishes a linkage between two logical records. The pointer field will contain the following information:

A. Type of file pointed to; whether it is keyed or non-keyed.

B. Type of data in the record pointed to:

   1) Pointer data only.
   2) Content data only.
   3) Pointer and content (pointer first).
   4) Content and pointer(s) (pointer information carried as a type of content, e.g., key itself encoded as the address).

C. The name of the file in which the record resides.

---

*Source: Compiled from computer runs made November 1 to November 6, 1967 against Santa Cruz catalog on SCAT 10, 11, 12. Compiler: Ralph M. Shoffner. Capital letters are counted as two characters. Note that none of these counts have been verified and they may contain errors.

FIGURE 5:

UNIQUENESS OF AUTHOR IDENTIFICATION

Number of Characters in Identification Tag

Number of Authors per Unique Tag	7	8	9	10	11	12	13	14	15	16
1	2866	3175	3635	4364	5399	6319	6895	7288	7496	7656
2	971	1030	1140	1290	1515	1628	1732	1765	1768	1770
3	492	518	544	605	655	683	699	692	678	670
4	333	335	347	358	343	346	323	322	324	323
5	184	183	176	200	201	203	204	201	194	190
6	148	150	152	152	164	142	134	128	128	127
7	98	92	96	89	91	88	82	81	82	79
8	60	63	63	62	65	55	56	55	51	51
9	78	73	73	66	58	47	47	42	41	39
10	49	50	45	46	42	38	36	32	30	30
11	39	42	29	34	26	21	19	15	15	15
12	27	28	30	29	23	18	21	18	18	16
13	29	26	25	26	23	26	21	23	22	21
14	24	21	22	19	19	13	11	10	9	11
15	20	19	18	18	11	12	13	14	13	11
16	16	18	14	13	11	10	7	6	8	6
17	8	11	11	14	12	12	12	12	9	9
18	9	8	9	13	13	9	7	5	5	5
19	18	16	14	13	9	9	6	5	5	5
20	9	9	9	12	8	9	9	7	7	7
21	14	10	11	9	6	2	1	1	1	1
22	10	7	9	8	9	4	4	4	4	4
23	7	8	8	3	2	2	2	2	2	2
24	4	5	4	5	1	1	0	0	0	0
25	8	6	5	5	3	1	2	2	2	2
26	5	4	3	3	4	5	4	3	3	3
27	3	3	4	5	2	2	2	2	2	2
28	5	5	3	3	3	1	3	2	2	2
29	6	6	5	3	3	2	1	1	0	0
30	7	4	3	1	1	0	0	0	0	0

Notes:

Table entries are the number of unique identification tags having a given number authors with the same identification tag (specified by the row value) for the given tag length (specified by the column value). The identification tag consists of the beginning characters of the author field in the order surname, forname, etc.

Source: 20,000 authors in alphabetical order
Compiler: Naresh Kripalani, March 6, 1968

FIGURE 6:

DISTRIBUTION OF NUMBER OF FIELDS OF LENGTH N

Length	Call No.	%	Author	%	Subject	%
1	1	0.0	0	0.0	0	0.0
2	0	0.0	16	0.0	0	0.0
3	0	0.0	36	0.1	4	0.1
4	0	0.0	33	0.1	65	0.1
5	0	0.0	37	0.1	172	0.2
6	5	0.0	50	0.1	462	0.5
7	14	0.0	136	0.2	924	1.1
8	79	0.1	64	0.1	1154	1.3
9	312	0.5	155	0.2	1197	1.4
10	1870	3.2	103	0.1	1434	1.6
11	4770	8.2	249	0.4	1662	1.9
12	9234	15.8	221	0.3	1676	1.9
13	12322	21.0	444	0.6	1651	1.9
14	6614	11.3	953	1.3	1778	2.0
15	4877	8.3	1399	2.0	1976	2.3
16	5180	8.8	1908	2.6	2072	2.4
17	4434	7.6	2339	3.3	2330	2.7
18	3717	6.4	2709	3.8	2346	2.7
19	2095	3.6	2740	3.8	2369	2.7
20	846	1.5	2759	3.9	2552	2.9
21	801	1.4	2698	3.8	2835	3.3
22	656	1.2	2768	3.9	3327	3.8
23	422	0.7	3014	4.2	3095	3.6
24	218	0.4	3165	4.4	3186	3.7
25	77	0.1	3145	4.4	3499	4.0
26	6	0.0	2972	4.2	2979	3.4
27	0	0.0	2837	4.0	2844	3.3
28	0	0.0	2664	3.7	3046	3.5
29	0	0.0	2745	3.9	3036	3.5
30	0	0.0	2826	4.1	2656	3.0
31	0	0.0	3145	4.4	2458	2.8
32	0	0.0	3311	4.6	2332	2.7
33	1	0.0	2965	4.2	2422	2.8
34	0	0.0	2713	3.9	2372	2.7
35	0	0.0	2277	3.2	2085	2.4
36	0	0.0	1919	2.7	1691	1.9
37	0	0.0	1542	2.2	1588	1.8
38	0	0.0	1230	1.7	1674	1.9
39	0	0.0	968	1.4	1314	1.5
40	0	0.0	969	1.4	1196	1.4
41	0	0.0	644	0.9	1224	1.4
42	0	0.0	573	0.8	1100	1.3
43	0	0.0	512	0.7	1033	1.2
44	0	0.0	472	0.7	1003	1.2
45	1	0.0	439	0.6	1042	1.2
46-56	4	0.0	2513	3.6	6355	7.2
Total	54,122	100.1	71,377	100.6	87,216	100.2

D. Length information: if a content record is pointed to, the number of characters in the record; if a pointer record (types 1 and 3), this element contains the number of fields in the pointer record.

In addition, if the pointer is to a non-keyed record, the pointer field must contain the record address; that is:

E. Relative block number, and

F. Relative position within the block.

If the pointer is to a keyed file the pointer record must contain:

G. The key value.*

The format of a pointer field is as follows:

POINTER TO A NON-KEYED RECORD: (NK)

A	B	C	D	F	E
T	D	File Name	Number/ Length	Pos.	BLOCK

0 1 2 4 6 8 12

POINTER TO A KEYED RECORD: (K)

A	B	C	D	G
T	D	File Name	Number/ Length	KEY

0 1 2 4 6 KL

The length of pointer NK is always 12 bytes; the length of pointer field K is always 6 plus the key length (KL) of records in the file to which this points.

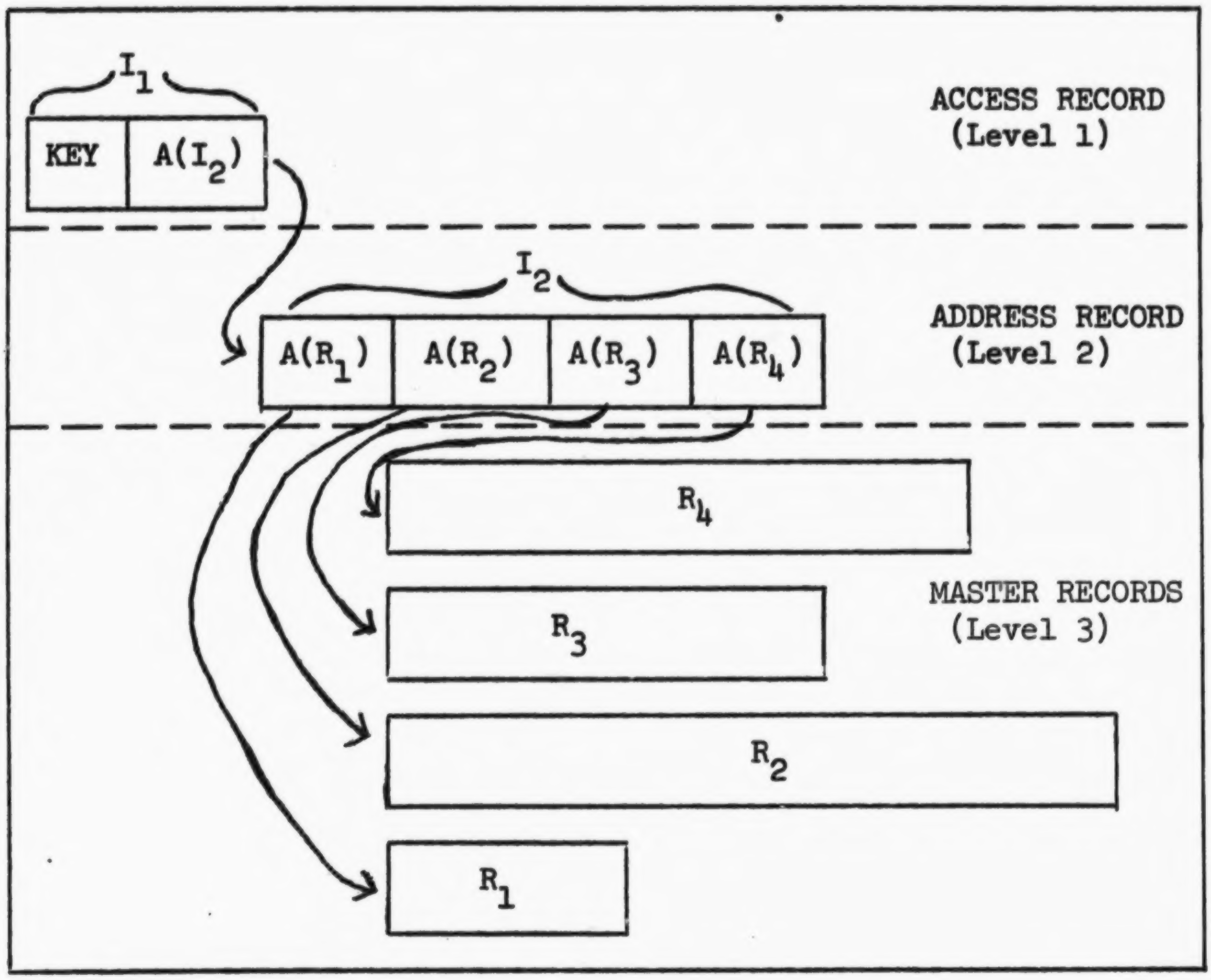
Thus in this arrangement we have a linkage established by the pointers. The access record points to the address record; the address record contains multiple pointers to master records. This linkage is shown schematically in Fig. 7.

---

*The key length for the file is obtained from an independent table called the file table.

FIGURE 7:

SCHEMATIC OF MULTI-LEVEL FILE STRUCTURE LINKAGE



I_1
KEY
$A(I_2)$
ACCESS RECORD (Level 1)
I_2
$A(R_1)$
$A(R_2)$
$A(R_3)$
$A(R_4)$
ADDRESS RECORD (Level 2)
R_4
R_3
MASTER RECORDS (Level 3)
R_2
R_1

Possible retrieval linkages are:

(1) I_1 only

(2) $I_1 + I_2$

(3) $I_1 + I_2$ + any combination of Master records:

e.g.: $R_1 + (R_2 \cdot R_3 \cdot R_4)$

Using the master and index files described above, it is possible to show the steps in a utility retrieval program. The program is supplied with the key name of an access point, along with a code indicating whether the name is of the author, title, or subject class. A search on key is performed in the indicated access file. If and when a match occurs, the data from the address is read into core storage.

At this point it is possible to provide the user with information which may be useful in satisfying the search request. It is possible, for example, to indicate how many master records have been indexed under the name, or to list the addresses of the master records. Otherwise, the processing continues and, using the addresses of the master record(s), the blocks containing them would be read into core storage, and the information transferred to the requestor at the on-line terminal.

III. THE BIBLIOGRAPHIC RECORD

A. GENERAL

Although record format and file structure are different problems, there are overriding considerations which caused us to devote considerable effort to the identification of record content and form in the first phase of the project. Among these reasons were three which are goals of the Project:

(1) To develop methods of converting machine files of bibliographic data both from manual catalog data and existing machine files;

(2) To devise techniques for processing the content of machine records via search routines, using actual data bases for realistic testing;

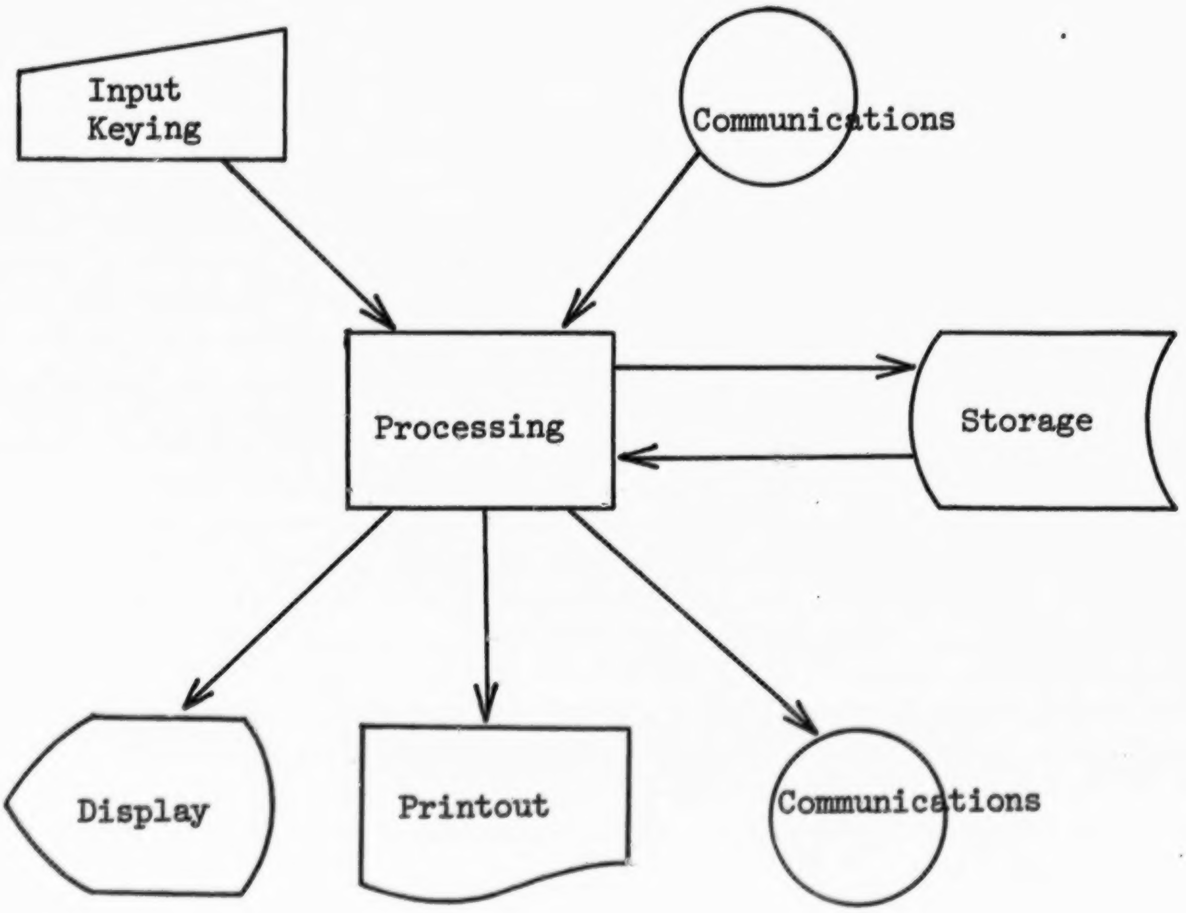
(3) To make the data base being established as a research vehicle for the Project, serve as a usable, available source of records both to other organizations and in practical applications, e.g., a computer-produced book catalog. This is particularly important in view of the large costs associated with the development of the data base.

Although we often refer to bibliographic records as if they were homogeneous objects, they vary greatly both in form and content. Particularly in a data processing system, records which refer to the same external item (such as a book) may assume radically different forms as they are transformed through the stages of input, processing, storage or output. Fig. 8 shows the major functional components of a data processing system, each of which may have different, often conflicting characteristics dictating a different desirable form of record, independent of its content. The use of a single record form throughout the system would mean that none of the conflicting requirements would be effectively met. In addition, the form of a record is very important to the file organization project if we are to meet our goal of providing a system within which experimentation can take place. A computer works entirely with the form of the data. For example, if it is to be possible to allow the author field of a record to contain either the author's name or a code standing for his name in an author authority index, the form of the records must be such that this difference can be signalled to the computer programs so that appropriate conversion can take place before processing or output. Thus, where a highly flexible system is the objective, record form and file structure are interdependent.

In a similar manner, we cannot divorce the record form from its content. That is, we wish to develop a system capable of handling records with content ranging from that of the traditional catalog card through those of index entries for journal articles,

FIGURE 8:

FUNCTIONAL SYSTEM COMPONENTS RELATED TO DIFFERENT RECORD FORMATS


Input Keying
Communications
Processing
Storage
Display
Printout
Communications

to those of augmented records containing abstracts, user comments and perhaps, full text. In order to assure that our system is properly designed, we must consider this range of content as we establish the various record forms.

In considering the form of the records to be handled, the questions must be broad and their answers must be quite precise if we are to achieve an open-ended facility that is capable of reasonably efficient operation even though in an experimental environment. Thus, we must determine not only how to tell what particular kind of record we have at hand or where a particular field begins and ends, but also which typographical symbols we have chosen to represent, how they are represented, and what will be done should we choose to represent other symbols at a later date.

The final part of this section is given over to a discussion of one aspect of bibliographical "closeness", that of the similarity of personal names. This is an extremely important type of similarity in the handling of bibliographic entries, in that the author is normally the major access point to a work. Our work is directed at overcoming errors due to unintended spelling variations in personal names.

B. RECORD CONTENT

The content selected for the data base does not constitute a radical departure from current bibliographic standards. That is, the major portion of the data base is anticipated to have the content of the traditional catalog card. In addition, some records will have additional index record and citation content. A few remarks will summarize the reasons for the decision to employ the conventional catalog record in a basically unmodified way. To reconstitute the basic information content of machine bibliographic records in advance of data on the use of current record content would change the focus of the project.* Our decision was to orient the building of a data base toward materials of general and current usefulness to the research library so that our research results could be of potential value at an early date. That goal in turn made it desirable that the content of these records be capable of representation in a standard format to facilitate interchange of data among libraries. The decision was made, therefore, to conform to the path being taken by the Library of Congress in its MARC Project, which is developing a standard bibliographical record content and format, beginning with monographs, and specifically intended for the data exchange function.

---

*For the opposite approach, see the reports from MIT's Project INTREX, in particular the Augmented Catalog Data Base.

Since the end of the MARC Pilot Project in June 1967, LC has moved through a phase of refinement and testing of the original MARC I record, preliminary to implementation of a machine cataloging distribution service to libraries. A "new model" of the format was issued in the form of specifications for the MARC II format in the Spring of 1968. Parallel to these developments, ILR had begun its File Organization Project in mid-1967.

The impact of this decision during the current phase of the study has been to place considerable stress on the project staff attempting to keep the Project work abreast of the MARC format development. We have found that the problems of coding using the MARC II format are of two sorts - matters of data availability and matters of difficulty of interpretation. The matters of data availability are those pertaining to the less predictably useful elements, e.g., to record whether the book contains an index to its own contents, or whether it is a festschrift. Without retrieving the book, editors coding catalog records cannot make such determinations. The other difficulty arises from interpretation problems, such as identifying sub-types of name headings in the MARC indicators, assigning of the language codes, identifying some of the sub-types of subject headings, determining when the main entry should be regarded as the publisher, and other areas where trained judgment and perhaps access to the book are mandatory for accurate data encoding.

At this time, most of these problems appear to be resolved. However, while accepting the defined content of the MARC record in general, there are some codes or elements, defined by LC, which we will not attempt to supply and there are others which we will supply which are undefined by LC, at this time. These content exceptions are listed briefly in Figures 9 and 10. It should be noted in Fig. 9 that a distinction is made between a data element and the corresponding code (e.g., field tag, sub-field delimiter, or whatever) which identifies the data element. For example, ILR will not supply the identifying code for Book Number (e.g., in the LC Call No. field) but the data will be present since it is embedded in the field. In most cases of exception to MARC II, ILR is deferring both the MARC II data element and its associated identifying code (if any), e.g., the Search Code, Tag 042.

C. RECORD FORM

1. The Need for Multiple Formats. We have defined for the File Organization Project an input record format and an internal processing format to accept the corresponding input data. The processing format in turn is convertible both to and from MARC II, that is, when the record is to be output for transmission of our locally-converted records on magnetic tape, or when LC-produced records are to be accepted into our data base. In addition, we are planning to develop a mass storage format using data compression techniques. The first question that is sometimes asked is "Why do you need many formats? Why not just one?"

FIGURE 9:

MARC II ELEMENTS DEFERRED IN FILE ORGANIZATION PROJECT DATA BASE

Data Element	Defer Identif. Code Only	Defer Both Data Element & Code
FIXED LENGTH DATA ELEMENTS FIELD		
Country of Publication		X
Intellectual Level		X
Festschrift Indicator		X
Index Indicator		X
Fiction Indicator		X
Biography Indicator		X
VARIABLE FIELD DATA ELEMENTS		
002 Legend Extension		X
014 Search Code		X
019 Local System No.	X	
020 BNB Classification No.		X
070 NAL Call Number		X
071 NAL Subject Category Number		X
080 UDC Number		X
220 Translated Title	X	
360 Converted Price		X
670 NAL Agric./Biol. Vocabulary		X
$ Book Number portion of call numbers	X	
$ Thickness		X

FIGURE 10:

FILE ORGANIZATION PROJECT DATA ELEMENTS NOT DEFINED IN MARC II

LEADER

Agency Code for Originator of Machine Record
Date of Machine Record Format Translation
Agency Code for Processor of Machine Record
Type of Source of the Catalog Data
Agency Code for Source
Agency Code for Adaptor of Catalog Data

FIXED LENGTH DATA ELEMENTS FIELD

Literary Group Filing Control
Cancel Title Added Entry Indicator

VARIABLE FIELD DATA ELEMENTS

052	Cataloging Source Legend
091	Copy Statement (Local Card) (Proposed)
570	"In Analytic" Note
580	"Full Name" Note
640	Book Title (as subject)
741	Title Added Entry (Periodical) (Proposed)
9--	NUC Card Number
9--	Superintendent of Doc's. Number
-12	Firm Name Heading (Proposed)
-31	Anonymous Classic Heading (Proposed)
$	Copy No. (in Holdings Field)

In many information system applications, only one record format is defined. For example, a library converting catalog data for any particular purpose, such as acquisitions searching or book catalog printing, might operate with essentially one format. The identifying codes used constitute a single structure applicable throughout the cycle of computer storage and retrieval.

In contrast to that approach, the Library of Congress and others are experimenting with multiple formats, viz., in the LC input record, mnemonic character groupings are used as acronyms to substitute for the field names (e.g., "MPS" stands for MAIN ENTRY of the type PERSONAL NAME, sub-type SURNAME). The internal processing codes, however, are the standard MARC II numeric tags, e.g., "100." Other libraries, such as Stanford and Toronto, are testing mnemonic notation for use as query tags, i.e., as identifiers used to interrogate the data base.*

It is our view that although the formats must be convertible one to another in order to perform functions such as input and search, the format requirements are dependent upon the particular function being performed. It would be cumbersome to impose the same notational solution, e.g., use of an identical mnemonic tag, throughout all the system's functions. Where large files of bibliographical records are being processed, the single format approach is also costly. Instead, workable formats must be constructed to meet the requirements of each function separately.**

The following are the functional requirements for record formats that we have identified.

(1) Input format.

(a) Ease of manual preparation and handling of the source data (selection, creation of coding sheets, etc.)

(b) Speed and accuracy in human editing of the source data. The codes should be easy to remember and simple and convenient to insert.

---

*See, for example: Bregzis, Ritvars. "Query Language for the Reactive Catalogue." In: Tonik, Albert B., ed. Information Retrieval: the User's Viewpoint - An Aid to Design. Philadelphia, International Information, Inc., 1967. pp. 77-91. (Fourth Annual National Colloquium on Information Retrieval, May 3-4, 1967).

**For a report on an approach to library file handling, with many of the record format features of which we find common agreement, see: Cox, N.S.M. and J.D. Dews. "The Newcastle File Handling System." In: Cox, Nigel S.M. and M.W. Grose, eds. Organization and Handling of Bibliographic Records by Computer. Hamden, Conn., Archon Books, 1967. pp. 1-21.

(c) As much streamlining as possible of the keyboarding. The keying should be rhythmic. Awkward strokes should be avoided.

(d) Independence of the format codes and the typographical character codes from the character set of the equipment, so that in principle any device can be used.

(2) Processing format.

(a) Facility in addressing individual data elements, grouping of data elements into segments, etc.

(b) Flexibility of structure, so that either fixed or variable fields can be efficiently handled by software.

(c) Repeatability of any data element and its format identifier.

(d) Versatility such that the format can encompass within a consistent structure, the varying content of records for all types of library materials: monographic, serials, journal articles, etc., in either conventional or augmented form.

(e) Minimization of character-by-character scanning, e.g., when preparing foreign language fields containing diacriticals for output display.

(f) Provision of capability for generation of sort keys with the minimum number of additional codes placed in the record.

(3) Mass storage format.

(a) Compactness of representation.

(b) Fast translation from and to the internal processing format.

(c) The effects of translation errors are localized or correctable.

(4) Communications format.

(a) Processable on computers with different memory and logic organizations.

(b) Low programming requirement to utilize the record.

There are many other requirements which the formats should meet. However, these will be sufficient to demonstrate the conflicting requirements from function to function. For example, there is a conflict between the compactness of the mass storage format and the facility in addressing individual data elements of the processing format. Also there is a conflict between the processing format versatility and the ease of coding the input format, since in general the fewer the things to be remembered the easier the coding. It is true that even in developing formats for the separate functions it is unlikely that our initial formats will remain unchanged throughout the study. However, regarded as a system of formats, they are likely to be more satisfactory than a single format would be.

Irrespective of the discrete functional demands placed on the sequence of formats, they all serve a basic purpose in common: they embody coding to identify record element content by "kind" and, where applicable, by the role played by the data contained as a value in an element. A name is an example of a kind of data element. A name may be one of several sub-types and can act in one of a number of roles in relation to a document. For each relationship it is assigned to a different field which identifies at once both its kind and its role. (E.g., kind = personal name; role = added entry, alternate author of document.)

In summary, the coding in the formats performs four important tasks. It supplies:

(1) Information about the document itself (title, subject headings, all the conventional bibliographic data).

(2) Information about the file (whether the record is new to the file, which file it belongs to, record status, etc.).

(3) Information about the record (to uniquely identify it, to characterize its composition, to distinguish it from other types of record content, etc.).

(4) Information about fields or about other codes (e.g., that a name in a field is of a certain sub-type; that a tag is to be processed under a certain condition, etc.).

2. Input Format. A summary of the approach underlying the development of the input format at ILR can be found in a previous Institute publication.* The present format is an extension and elaboration of work reported therein.

a. Background. The input format described here, and the conversion procedure outlined below (in Section IV) represent an approach chosen in the earliest stage of the data base preparation task. The initial work has resulted in a draft coding manual for use in manual editing.* We anticipate that the input format and the editing procedure will evolve towards greater reliance on computer-assisted field identification. As our experience is evaluated, conclusions will be drawn about the feasibility of placing more emphasis on such computer-set codes. The ultimate, of course, would be optical scanning of the unedited cards or a straightforward keyboarding of them. There would be no intervention by a human editor until after processing by a computer program. This program would format the record and then print out the results for post-edit inspection by a human and correction as needed. Corrections to errors in coding committed by the edit program algorithms or the input device operator would be input to the machine file in the normal fashion. We are currently working toward this goal with the automatic translation of the Santa Cruz records. These records have only the most general level of field coding.

---

*Cartwright, Kelley L. and R.M. Shoffner. Catalogs in Book Form: A Research Study of their Implications for the California State Library and the California Union Catalog, with a Design for their Implementation. Berkeley, Institute of Library Research, University of California, 1967. pp. 30-35.

b. Coding scheme. Since most of the catalog data is variable in length, the bulk of the logical fields have been defined as variable for the input format as well as the processing format. The advantage to this is that in conversion from 3x5 cards a more normal text typing rhythm is permitted than if a fixed length field approach were used.

More than 120 data elements and fields have been identified in the MARC II format, and equivalent ILR input tags have been assigned for most of them. These are translated into the MARC II tags and indicators in the ILR processing format.

The manner of identifying the variable fields for input is a key feature of this format. A mixed technique has been employed: 1) insertion of signals which explicitly label the beginning and end of each field, and 2) depending on the predictability of the occurrence of fields, the insertion of an identifying tag. The first is a uniform symbol for fields which are always or nearly always present, and which always occur in the same sequence. The second is a unique symbol applied to those fields which occur infrequently or in isolated sequences.

The first type of coding is applied to the body fields (author, title, etc.). In this case, no field code may be omitted, even though the field contains no value in a particular record. The "blank" field is purged from the input record at the time of its transformation into the storage record. Ten fields were chosen for coding by this technique on the basis of previous experience and sampling. A symbol easily insertable in the dense text of the catalog card was selected: the slash mark, "/", is used to signal the beginning of each of the "body" fields. A blank field, e.g., absence of a publisher name, would be signalled by two slashes, "//".

---

*Cunningham, Jay L. Instruction Manual for Editorial Preparation of Catalog Source Data. Preliminary Edition. Berkeley, Institute of Library Research, University of California, 1968. 172 p. Because this manual is a draft which is expected to be revised it has been provided only in limited quantity.

The second type of code is two-character tag composed of a uniform special character plus a unique alphabetic. When a field is not present, no tag is inserted. Two series of these codes were defined. The more frequently occurring fields are coded with a combined asterisk plus lower case alphabetic (for ease of input typing). The less frequently found fields are identified by an exclamation point plus a single character alphabetic. The "*" or "!" is needed to distinctly identify the tag, otherwise the edit program could not distinguish reliably between the code letter and a text letter.

A further conciseness in input coding derives from the convention that the editor need write only the lower case alphabetic code on the sheet, in the case of the "asterisk" series of tags. For distinctiveness, the code letter is written in red. The input device operator must recognize the letters written as tags and preface each "red letter" keyed with an asterisk. For the other series of tags, the editor writes out both the exclamation point and the code letter.

One problem with the brevity inherent in this system is its lack of mnemonic value. The input tags used in the LC MARC production system, for example, are more easily remembered than combinations such as asterisk plus a letter. However, we have found that a simple checklist supplied to the editor as a ready reference tool suffices for the majority of tags not quickly learned through repetitive use.

c. Coding sheet. Information will be input to the computer from a coding sheet on which a catalog card is reproduced or attached. The principal reasons for the use of the coding sheet are: (1) it provides space in which to record certain information which is not explicit on a catalog card; (2) it provides space in which the editor of the data can write information which represents additional data required, or modification of information already on the card; and (3) it provides checklist reminders of some of the coding conventions and options.

In Fig. 11 we present an example of the coding sheet developed for the project. The fixed field alternatives and certain other codes are printed at the left and at the bottom of the sheet. The tags and field names used in the ILR input format are listed in Appendix V.

3. Processing Format. The details of the processing format for the File Organization Project data base are described in Appendix IV. The salient features, and certain significant

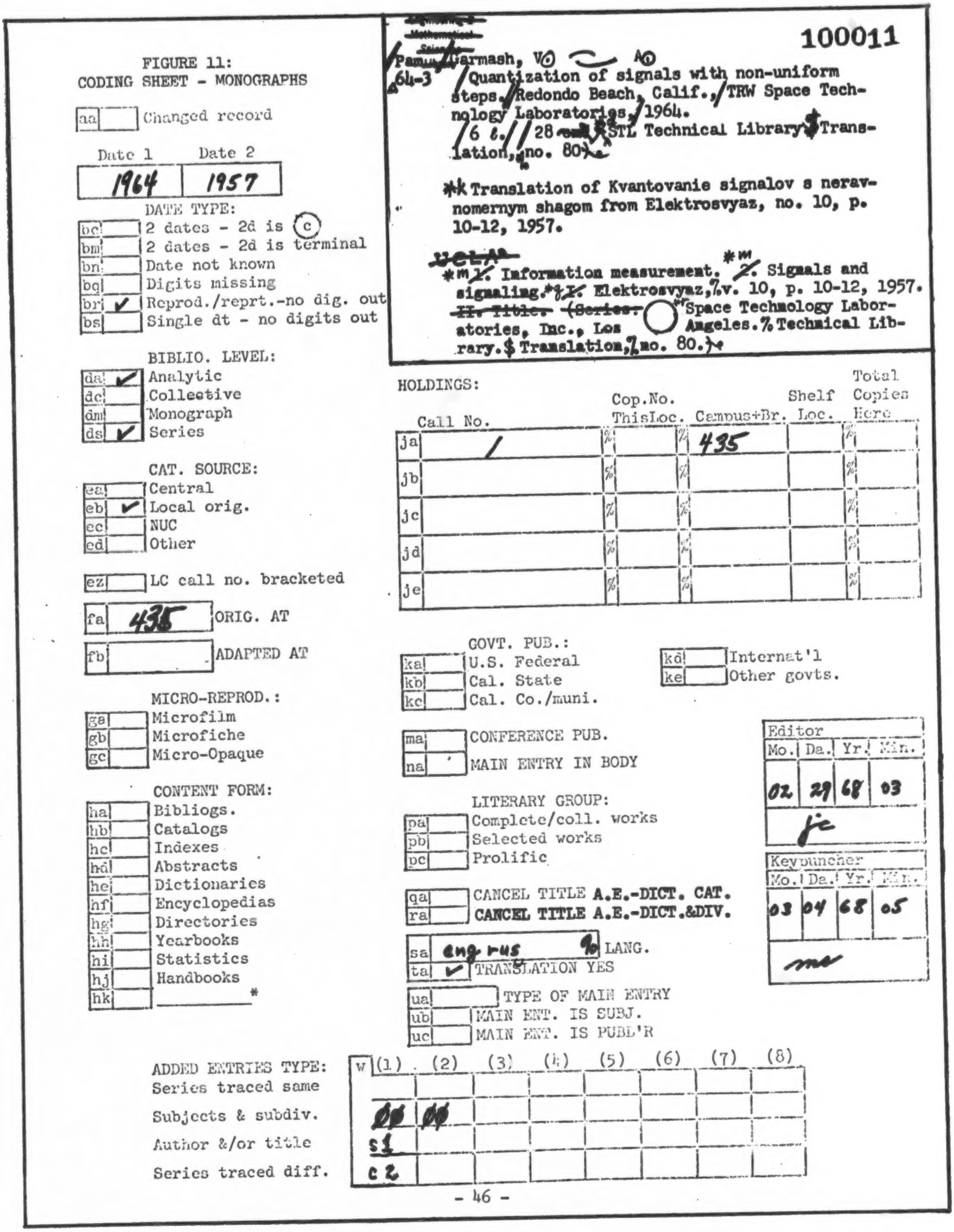


FIGURE 11:
CODING SHEET - MONOGRAPHS

aa ☐ Changed record

Date 1	Date 2
1964	1957

DATE TYPE:
bc ☐ 2 dates - 2d is ⓒ
bm ☐ 2 dates - 2d is terminal
bn ☐ Date not known
bq ☐ Digits missing
br ☑ Reprod./reprt.-no dig. out
bs ☐ Single dt - no digits out

BIBLIO. LEVEL:
da ☑ Analytic
dc ☐ Collective
dm ☐ Monograph
ds ☑ Series

CAT. SOURCE:
ea ☐ Central
eb ☑ Local orig.
ec ☐ NUC
ed ☐ Other

ez ☐ LC call no. bracketed

fa 435 ORIG. AT

fb ☐ ADAPTED AT

MICRO-REPROD.:
ga ☐ Microfilm
gb ☐ Microfiche
gc ☐ Micro-Opaque

CONTENT FORM:
ha ☐ Bibliogs.
hb ☐ Catalogs
hc ☐ Indexes
hd ☐ Abstracts
he ☐ Dictionaries
hf ☐ Encyclopedias
hg ☐ Directories
hh ☐ Yearbooks
hi ☐ Statistics
hj ☐ Handbooks
hk ☐ ________*

~~Engineering & Mathematical Sciences~~

100011

~~Pamu~~/Garmash, V⊙ ⁀ A⊙
64-3 /Quantization of signals with non-uniform steps./Redondo Beach, Calif.,/TRW Space Technology Laboratories,/1964.
/6 l./ /28 ~~cm~~ STL Technical Library $Translation, no. 80)

*k Translation of Kvantovanie signalov s neravnomernym shagom from Elektrosvyaz, no. 10, p. 10-12, 1957.

~~UCLA~~
*m ~~1.~~ Information measurement. *m ~~2.~~ Signals and signaling. *q ~~I.~~ Elektrosvyaz, %v. 10, p. 10-12, 1957. ~~II. Title.~~ ~~(Series:~~ ○ *r Space Technology Laboratories, Inc., Los Angeles. % Technical Library. $ Translation, % no. 80.)

HOLDINGS:

	Call No.	Cop.No. ThisLoc.	Campus+Br.	Shelf Loc.	Total Copies Here
ja	/	%	% 435		%
jb		%	%		%
jc		%	%		%
jd		%	%		%
je		%	%		%

GOVT. PUB.:
ka ☐ U.S. Federal
kb ☐ Cal. State
kc ☐ Cal. Co./muni.
kd ☐ Internat'l
ke ☐ Other govts.

ma ☐ CONFERENCE PUB.
na ☐ MAIN ENTRY IN BODY

LITERARY GROUP:
pa ☐ Complete/coll. works
pb ☐ Selected works
pc ☐ Prolific

qa ☐ CANCEL TITLE **A.E.-DICT. CAT.**
ra ☐ **CANCEL TITLE A.E.-DICT.&DIV.**

sa eng rus % LANG.
ta ☑ TRANSLATION YES

ua ☐ TYPE OF MAIN ENTRY
ub ☐ MAIN ENT. IS SUBJ.
uc ☐ MAIN ENT. IS PUBL'R

Editor			
Mo.	Da.	Yr.	Min.
02	29	68	03
jc			

Keypuncher			
Mo.	Da.	Yr.	Min.
03	04	68	05
me			

ADDED ENTRIES TYPE:	w (1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Series traced same								
Subjects & subdiv.	00	00						
Author &/or title	s1							
Series traced diff.	c2							

differences in form and structure of this format in relation to the input format on the one hand, and the MARC II Communications Format on the other hand, are outlined briefly below.

a. Input and processing format differences. There are two main differences between the input and storage formats:

(1) Default. Certain data elements and codes are set by default in the edit program and thus do not appear on the coding sheet or in the external input record. An example is "Type of Record = a-- language material, printed."

(2) Coding notation. A briefer but flexible form of field coding (as described above) is used for input rather than three or four character numeric tags as in the MARC system. The three-digit MARC II tags and associated codes are set by the INFOCAL edit program, which translates the input codes into corresponding internal processing format codes, as it compiles each field by concatenating the various input codes and elements that make up the internal record. The input record image is, in general, not a mirror image (identified by different field codes) of the resultant internal record in processing format. This is because of 1) the default values, mentioned above, 2) separation of parts of certain tags and elements on the coding sheet for purposes of editor convenience, which are brought together by the edit program for the internal record, and 3) the more complex structure of the internal coding, due to the fact that a record directory method is used to organize the processing record.

b. Processing format and MARC II differences.

(1) Figures 9 and 10 in Section III.B. listed the few minor differences between the File Organization record content and that of the MARC II record. These differences were primarily those of inclusion/exclusion. The great majority of data elements that constitute the content of the File Organization Project record will be identical to those in the MARC II record.

(2) There are very few differences between the File Organization record and MARC II in terms of coding - i.e., the same field and sub-field tags, delimiters and indicators which identify and characterize the data element content. In a few cases ILR will not be able to supply a code for a data element identified in MARC II, and in a few other cases, we elected to go slightly beyond the Library of Congress in identifying a field to serve special requirements.

(3) In respect to record structure, the File Organization record is quite similar to MARC II - a fixed length Leader is created, with quite similar purpose and content as MARC II. A Record directory controls the access to the remainder of the record, including the Fixed Length Data Elements field, which is treated for programming purposes as if it were a variable field length field.

In all cases, our goal has been to conform to the utmost extent to the MARC II design, in order to minimize software differences in routines to handle records of our own creation and records originating elsewhere, e.g., in the LC distribution service. In this regard, we expect to do no reformatting of the content of MARC II records received from LC. A certain minimum addition of data will be necessary, e.g., a local master record number, local call numbers, etc. Lastly, a certain amount of reformatting of record structure is planned, e.g., to move the codes for diacritical marks from the text of the field to a special field header. This will facilitate preparation of the field for display on CRT terminals having limited character sets and allow us to begin experiments in which the very highest quality of display of foreign language text in bibliographic records is not a significant factor.

These differences can be reviewed in more detail in the processing format specification in Appendix IV.

In summary, these differences in formats are not considered to be departures from the concept of a standardized bibliographic record. The MARC II design is specifically intended to be adaptable to local needs, and hospitable to non-conventional data and coding. Standardization does not imply a rigid, uniform set of data elements constituting a monolithic structure. Instead, through the use of the mechanism of optional tags and fields, modularity of structure, and a hierarchical coding definition, libraries can comply with MARC yet tailor it to specific requirements. The acceptance of this approach is already perceptible at the national level in the efforts of the three U.S. national libraries to support MARC II as a common standard by accommodating certain of their elements to a common definition, while defining other elements and codes for their own particular applications.

D. THE REPRESENTATION OF TYPOGRAPHICAL CHARACTERS*

1. Overview. Once the input format and the storage format for bibliographic records have been designed, the problem of keyboarding textual material not represented on the keyboards of standard devices must be faced. Catalog records contain tremendous linguistic variety, because certain portions of the data (usually at least the title) are by convention recorded in the language of the text or its translation. Thus the conversion system must embody procedures for keying both non-Roman alphabets and the diacritical marks, symbols, and other special characters, some of which may occur in any alphabet. Moreover, provisions must be made for efficient representation of such characters when they are not defined in the internal operating code set of the particular computer used.

---

*This section was prepared by Thomas Hargrove, of the ILR staff.

In line with the goal of minimization of the overall cost of conversion, the basic approach taken was not to secure specially modified input transcription devices which would contain a limited number of symbols and special characters beyond the standard set. It was felt that this approach is too costly at this time and represents only a proximate solution to both the input and the storage of special catalog data. Rather, a more universal concept was developed, wherein the standard keyboard of a readily available keypunch or other input transcription device could be used. This would require only a small amount of extra operator training and would be virtually unlimited in its potential for expansion to cover symbols and codes not foreseen at the beginning of the conversion.

2. General Objectives. The objectives of the physical data representation technique are:

a. Input device independence. To handle the extensive set of alphabets and characters, a method must be established to represent those characters by a notation that can be implemented on any equipment.

b. Compact internal character set. The codes used on the input device must be transformed into a concise internal set. This will be based upon the specific arrangement of the computer upon which the system is being implemented, but need not be logically tied to a given machine.

c. Standard interchange code. For data exchange purposes, such an internal code set could be translated to the ASCII extended character set being proposed for standard library use by the Library of Congress.

d. Ease and economy of coding. The technique developed at ILR assumes two aspects to the representation problem: conversion of individual special characters, and conversion of special alphabet streams (e.g., text in Cyrillic). At input, it is desirable to have a mnemonic code for those individual characters not on the particular keyboard employed. Yet it is desirable also to have concise codes. For a special character sequence, a control or escape code is needed to show the beginning and ending of the sequence and a shift to the original or to another sequence.

Each of these requirements poses special difficulties in the design of an economical special character coding technique.

e. Transliteration problem. The conversion of special alphabets (e.g., Cyrillic) could probably be efficiently handled by having special masks placed on the device keyboard and trained language operators type the foreign language records using the special alphabet control signal. The entire record would be stored in an internal configuration translatable to the external display repertory desired. For example, on a CRT device, the record stored in the coded equivalent of Cyrillic could be transformed automatically through a table for display in transliterated or Romanized form on the screen. A printed transliteration table would be available at the console for the user to convert from the transliterated form into his own language. It is recognized that this approach is fraught with several dangers since there is no unified agreement in the linguistic world on standard transliteration or Romanization schemes.

A remaining task is to establish the precise mapping of the input character codes into the particular computer character set available in the Project facility. Numerous smaller problems have also been identified, such as case change, spacing, and a signal that a character used as a format code has occurred in the character stream as text and should be so treated.

The remainder of thissection summarizes the rules for input keying of either individual special characters or of character sequences.

3. Basic Assumptions. The assumption underlying the proposed input codes is that all special characters can be keyed from a physical point of view. Such an approach might be called the "printer's" point of view, in contrast to a computer point of view of subsequent encoding of characters economically in storage, or to a user's point of view of the semantic or logical meaning which the characters serve to express.

A printer, without knowing the subject, is able to set characters by knowing only their physical sequence and alignment in text. A keying operator, similarly, should be able to keyboard codes for the physical shapes and alignment of the same characters. In this approach, the codes typed would not depend on the operator's having to know the meaning, or how the character would be coded in computer storage. Instead, the codes should be those which make input easiest for the keying operator, with the highest accuracy.

This proposal considers all special characters as established shapes, in a given horizontal position, with a defined vertical location, and physically imposed upon one or more textual positions.

The approach is aimed at providing the benefits of:

a. short codes for the most frequent characters;

b. minimal look-up for less frequent characters by use of combined codes of basic shape+location+applicability, in lieu of looking up arbitrary, non-structural codes;

c. method for adding as many codes as needed;

d. memorizing by keying operator of most frequent character codes plus the few basic qualifying codes.

4. Special Character Encoding Situations. For the purposes of input a special character representation scheme must, in principle, address five separate but interrelated situations that may occur in textual material:

a. Special (non-Roman) alphabets: e.g., Cyrillic and any other alphabet not represented on standard input devices.

b. Special marks: within any given alphabet, e.g., diacritical marks (acute/grave accent, etc., and combinations of these, digraphs, etc.)

c. Font change: within any given alphabet, font can be either significant or aesthetic (for emphasis). Significant font change is one involving semantic change, e.g., bold face to denote vector quantities in mathematical notation, italics to denote variables, etc. Aesthetic font changes usually add emphasis (e.g., italicized words), provide reader aids (e.g., boldface subheadings in text), or have other not strictly semantic purposes. The latter have been excluded from the technique devised, but could be handled in principle.

d. Special symbols: characters not on the standard keyboard, which may or may not have conventional meanings. They may be language-dependent or context-dependent for their meaning (e.g., £ in British money; ‡ has a special meaning on an LC catalog card.)

e. Special positional configuration: of any character in any language. E.g., exponents and subscripts have a conventional position and may have smaller point size in a string of text.

Several of the above conditions may apply to a given character. The single special character in a stream of text may be considered as a special case of the problem of handling the entire stream in a non-keyboardable alphabet.

To devise a systematic coding scheme to cope with these text situations, we redefined them into the following categories:

(1) The alphabet can be either:

Regular mode - the Roman alphabet as represented on the keyboard of the particular device used, is regarded as the nominal or default case; or

Special mode - alphabets other than Roman, e.g., Greek, Cyrillic, etc., not represented on the keyboard of the device used.

(2) A character stream in a given alphabet can be either:
Unspecified length; or
Specified length.

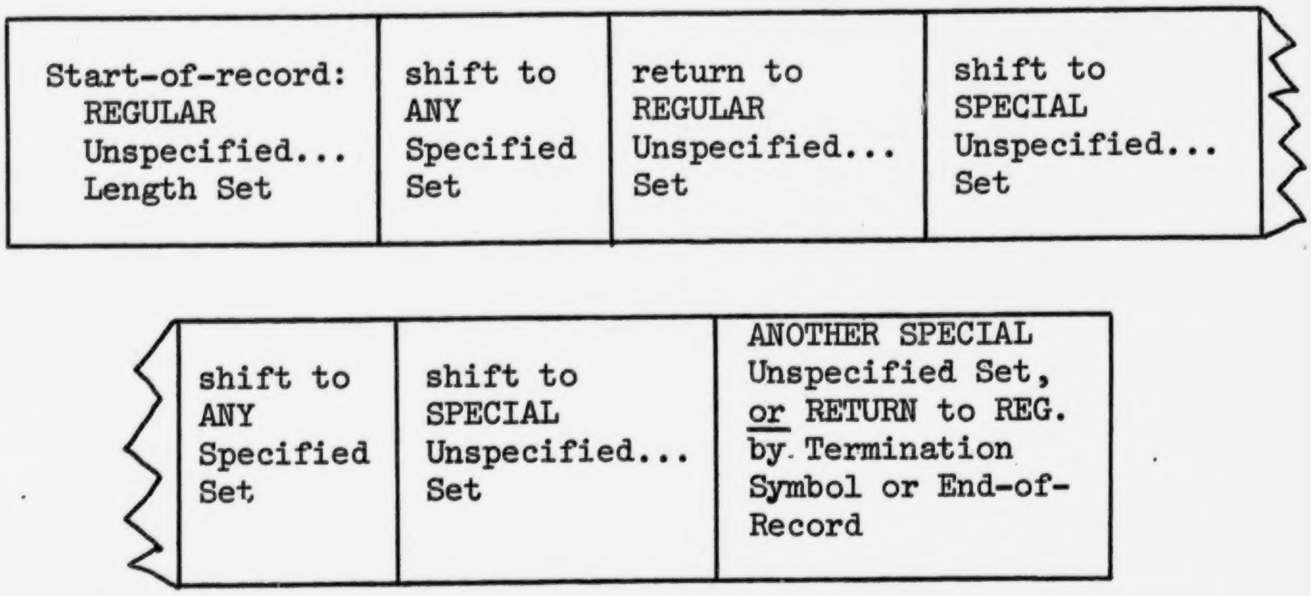
(3) A specified length character stream is initiated by a shift code signalling either a:

Fixed length code (predefined) standing for a special character; or a

Variable length code (e.g., abbreviated words predefined in a program table).

Breaking the textual situations down in this manner allows us to think how the keying might be best performed in a continuous fashion with the least stop-and-go motions by the device operator. The data can be regarded as contiguous blocks or strings within a character stream. Unit codes are applied by the operator to control each block that departs from the default situation ("Regular mode"). To illustrate, a catalog record in which a number of special text situations occur can be keyed as a series of contiguous blocks forming the total record:

FIGURE 12: KEYING BLOCKS OF TEXT


Start-of-record: REGULAR Unspecified... Length Set
shift to ANY Specified Set
return to REGULAR Unspecified... Set
shift to SPECIAL Unspecified... Set
shift to ANY Specified Set
shift to SPECIAL Unspecified... Set
ANOTHER SPECIAL Unspecified Set, or RETURN to REG. by Termination Symbol or End-of-Record

Specifically, the text categories defined above would be applied as follows:

(1) For normal Roman alphabet text not modified or interrupted by change in alphabet, by diacritical marks or by any characters not present on the keyboard, no special action need be taken by the device operator. Each record would be assumed to start its text in "Regular" mode.

(2) When any of the three categories of text situation occurs, as a departure from the Regular mode, the device operator can still continuously input data, in any alphabet, with or without diacritic marks or special characters, merely by input of a shift code when

departing from the normal situation. The "Regular" alphabet defined for input keying may be somewhat expanded for storage purposes. For example, the S/360 internal set has 256 possible codes, of which a number are already designated for graphics not present on the standard keypunch. It would not be necessary to store the input codes in the exact form in which they were keyed. Moreover, the convertibility is three-way: the input codes may be transformed to the most convenient and efficient storage codes, and the storage codes may be again transformed to available output devices, e.g., printers.

While keying in "Regular" mode, each character represents itself exactly as denoted on the keyboard. At the point of interruption, subsequent characters stand for either a character in another alphabet, a diacritical mark, or some other special character not denoted on the keyboard.

Two types of departures from the Regular mode can occur: Unspecified-length shift to Special mode, followed at some finite point by either a deshift to Regular or a shift to another Special set.

For example, a book title might have a Greek word embedded in an English phrase, followed by an alternate title in Cyrillic.

The deshift would be signalled by either a terminator code or automatically by the end-of-record mark input by the device operator, if appropriate.

Specified-length shift codes can be of two sub-types:

1) Fixed length codes, which are either one or two-character format codes recognized by the input edit program as field signals, delimiters, etc., or are these same characters themselves preceded by a reserved symbol to show that the character is in this case to be regarded as text data, not as a code.

Other than these occurrences, a number of doublet codes have been provisionally defined to represent diacriticals, e.g., "¢E" for acute accent. Triplets, etc. could also be specified, in order to make the system expansible.

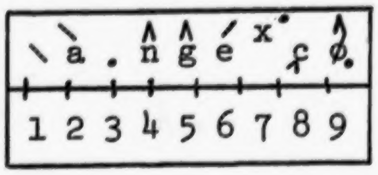
2) The second sub-type is variable-length specified shifts, as in the case of groups of characters predefined in a program table to represent abbreviated words, either for the purpose of economy of keying frequently occurring words in text, or for compact storage, or both.

5. The Identification of Shape, Size, Attitude, and Relative Position. Diacritics in particular manifest many combinations of shape, point size, inclination or attitude, and position relative to the character they modify. For brevity, we shall discuss these characteristics using the single term "location".

a. Order and location of characters. Special characters or Special uses of any character are considered for keying purposes to be in the same position or in juxtaposed positions of the data, left to right.

Data

Position No.



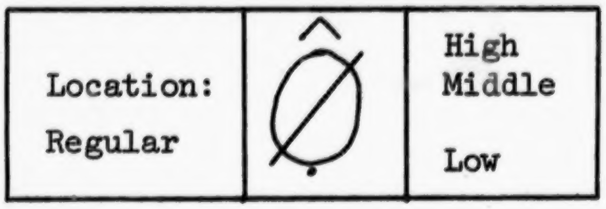
For purposes of input, all special characters are coded in relation to other characters by both physical shape and horizontal and vertical locations. The same shape in different inclinations is considered a different shape, for coding purposes:

¿ differs from ?   à differs from a̍ differs from á

The same shape character in different horizontal and vertical locations is coded to express both uniqueness of shapes and position in relation to a character to which it is either attached or juxtaposed.

	Before	With	After	character
High	^{-}d	$\bar{d}$	d^{-}	
Middle	-d	đ	d-	
Low	$_{-}d$	$\underline{d}$	d_{-}	

b. All characters including diacritic marks at the same position are considered vertically in a regular location, or in high, middle or low locations (drawn large for illustration):



(Circumflex is High)
(Slash is through Middle)
(Letter O is Regular)
(Dot, or period, is Low)

(1) Alphabets, digits, punctuation, and additional characters, such as +, ‡, are at the Regular location.

(2) Diacritics and special uses of any character, such as exponents and subscripts, are at High, Middle, or Low locations.

6. Initial Codes for Provisional Implementation. A double-character code set for modern European characters was defined using the ten digits to represent the most frequent diacritical marks. The most frequent letters specially formed with diacritical marks, and other separate diacritical marks would be represented by the letter codes. This double-character code set would serve as an extension of the Regular character set. By being fixed-length, the diacritical mark codes can be used in the midst of any special alphabet set.

The location of a diacritic in relation to a letter is implied by its code. The implication of this is that any special character which exactly coincides with a Roman letter and a diacritic mark may be coded either as a letter plus a diacritic mark, or as a single combined code. The frequency of certain diacritically marked letters in certain languages might make single combined codes very useful. However, for the present, the letter-plus-diacritical mark code is proposed as a scheme applicable to any particular language.

Unfortunately, a 36-character double code set is not enough to encompass all the diacritical marks. LC lists at least 64 special characters beyond the normal keyboard, in its ASCII extension proposal. These could be assigned single code characters in two separate sets, but for simplicity a two-character code is assigned provisionally to the less frequent diacritics. The order of the LC list does not reflect the frequency of occurrence of the marks themselves. LC's Information Systems Office is making a statistical study of the frequency of occurrence of diacritics, and when this kind of data becomes available, it will guide the refinement of the system proposed. In advance of such quantitative design information, we have used the diacritic set proposed by Palmer (see Fig. 13), and our own research to guide the establishment of the initial set (see Fig. 14).

When the frequency distribution of the diacritics and other symbols is better established, we will implement the LC list as needed, to assure full convertibility of the textual content of the data base to a MARC II character set standard.

Codes for special characters and for special uses are made up of a combination of a flag plus "¢" plus digit(s) or letter(s).

The most frequent codes have defined shape, location, and number of positions applicable in one code. Diacritic characters, for example, occurring most frequently are coded with briefest codes (given vertical position assumed in each case as the default position). E.g.:

acute (high, one-position) is coded ¢A
micron (breve, short)(high, one-position) is coded ¢B
cedilla (low, ") is coded ¢C

FIGURE 13:

THE TENTATIVE HARVARD LIST OF DIACRITICS

ILR 029 Codes	Harvard List of Diacritics*	Comments
¢E	´ acute	
¢A	` grave	
¢F	^ circumflex	
¢U	¨ dieresis or umlaut	
¢N	~ tilde	
¢C	¸ cedilla	
¢K	ˇ hacek	
¢G	° (as in Swedish å, Czech ů)	Angstrom
¢19	_ underline	Digit= No. of letters underlined.
¢O (letter O)	/ (as in Danish Ø)	**Display size adapts to letter
-¢M	- (as in Đ)	Anglo-Saxon Đ differs
¢O (letter O)	⁄ (for Polish ł)	**Display size adapts to letter
,¢L	, (comma under letter as in Rumanian)	
.¢L	. (dot below letter)	
.¢H	· (dot above letter)	
¢1W	˛ (used in Polish & Lithuanian)	Inverted cedilla
¢2W	″ (used in Hungarian)	
¢R	¯ macron	
¢B	˘ micron	

Note: Unique letters after ¢ were chosen as non-shift, and to coincide with frequently associated letters where possible, with allowance for specially reserved symbols.

*Source: Palmer, Foster M. "Conversion of Existing Records in Large Libraries; with Special Reference to the Widener Library Shelflist." In: Harrison, John and Peter Laslett, eds. The Brasenose Conference on the Automation of Libraries. Held at Oxford, Eng., 30 June-3 July 1966. London, Mansell, 1967. p. 74.

**In display the slash through "ł" of Polish font will be shorter than the larger slash through "ø" of Scandinavian font.

Slash, as cross-over, can be keyed as @/ in conjunction with the ¢T9 code for multiple applicability.

FIGURE 14:

ALPHABETICAL INDEX OF DIACRITIC CODES

Code	Mark	Name, Defined Loc.
¢A	`	Grave (high)
¢B	˘	Breve, micron (high)
¢C	¸	Cedilla (low)
¢D9	œ	Diagraph, up-to-9
¢E	´	Acute (high)
¢F	^	Circumflex (high)
¢G	•	Angstrom (high)
¢H		Undefined <u>high</u> location
¢I9	_	Underline, up-to-9 (low)
¢J9	⁀	Ligature, up-to-9 (high)
¢K	ˇ	Caret, Hacek (high)
¢L		Undefined <u>low</u> location
¢M		Undefined <u>middle</u> location
¢N	~	Tilde (high)
¢.9		Specified number to 9 of non-keyable characters
¢O	/	Slash (through middle)
¢P		Slant-in-font series (Italic)
¢Q		Weight-in-font series (bold)
¢R	¯	Macron (high)
¢S		Special alphabet shift
¢T9		Multiple-applicability (to 9)
¢U	¨	Umlaut, dieresis (high)
¢V		Reserved-further coding
¢1W	˛	Inverted Cedilla (low)
¢2W	˝	Double acute (high)
¢3W	ʽ	Reverse comma (high)
¢4W	=	Double underscore (low)
¢5W	ँ	Candrabinde (high)
¢6W	Ð	Demi-bar in Ang. Sax. (middle)
¢XHML		High or middle or low illegible
¢..		Unspecified number of non-keyable characters

FIGURE 14 (Cont.):

ALPHABETICAL INDEX OF DIACRITIC CODES

Code	Mark	Name, Defined Loc.
¢1Y	†	Dagger
¢2Y	‡	Double dagger
¢3Y	£	Pound sign (English)
¢4Y	¿	Inverted question
¢5Y	¡	Inverted exclamation
¢6Y	þ	Thorn (Icelandic)
¢7Y	b	Musical flat
¢8Y	ı	"Undotted i"
¢9Y	‘	Single left quote (high)
¢10Y	“	Double left quote (high)
¢11Y	”	Double right quote (high)
¢12Y	«	Left Cont. quote
¢13Y	»	Right Cont. quote
¢Z		Astride symbol
¢¢		Termination of special alphabet (de-shift)

FIGURE 15: PROPOSED SINGLE KEYING CODES COMPATIBLE WITH TRANSLITERATION SCHEMES FOR MODERN CYRILLIC*

Upper Case	Russian Letter Cap. Ital.	Applied Mechanics Reviews	U.S. Library of Congress	Journal of Symbolic Logic	Lower Case
A	А *а*	a	a	a	a
B	Б *б*	b	b	b	b
V	В *в*	v	v	v	v
G	Г *г*	g	g	g	g
D	Д *д*	d	d	d	d
E	Е *е*	e	e	é	e
2	Ё *ё*	ë	ë	ë	2
X	Ж *ж*	zh	z͡h	ž	x
Z	З *з*	z	z	z	z
I	И *и*	i	i	i	i
J	Й *й*	ĭ	ĭ	j	j
K	К *к*	k	k	k	k
L	Л *л*	l	l	l	l
M	М *м*	m	m	m	m
N	Н *н*	n	n	n	n
O	О *о*	o	o	o	o
P	П *п*	p	p	p	p
R	Р *р*	r	r	r	r
S	С *с*	s	s	s	s
T	Т *т*	t	t	t	t
U	У *у*	u	u	u	u
F	Ф *ф*	f	f	f	f
H	Х *х*	kh	k͡h	h	h
C	Ц *ц*	ts	t͡s	c	c
3	Ч *ч*	ch	c͡h	č	3
W	Ш *ш*	sh	s͡h	š	w
4	Щ *щ*	shch	s͡hch	šč	4
5	Ъ *ъ*	″	″		5
Y	Ы *ы*	ï	y	y	y
6	Ь *ь*	′	′	′	6
7	Э *э*	é	é	e	7
8	Ю *ю*	yu	i͡u	ú	8
9	Я *я*	ya	i͡a	á	9
Q one 1 zero 0	Ѣ ѣ I i Θ ѳ		i͡e ī ḟ		q 1 one o zero

CAPITAL letters keyed with Underscore (0-5-8) before code.
Difference in fonts implied in code for Character Set.
Other characters (such as V, transliterated by L.C. as ẏ) can be coded as elements of another character set, acting as a miscellaneous, overflow, or Slavic character set containing all Cyrillic characters not already coded in the Russian character set.

---

*Source: Mathematical Reviews, (Am. Math. Soc., Lancaster, Pa.) v. 30 (1965), p. 1207.

The code for the diacritical would be keyed following the character which it modified or with which it is associated. Special conditions for which this rule is ambiguous and requires further specifications are provided for in the keypunching manual.

7. Particular Scheme for Cyrillic Alphabet. For Cyrillic, it is proposed that a 36-character code set (26 Roman alphabet and 10 digits) to be used for keying and storage is more convenient, economical and unique for operator recognition purposes than transliteration schemes of a linguistically controversial nature. With unique codes stored for every Cyrillic character, printout can be made into Cyrillic, or into any transliteration scheme desired. What is left undefined is a complete transliteration scheme. It is emphasized that it is the alphabet (made up of graphemic units) which is being coded, not the "language" in transliteration. A Russian word that already appears in Romanized character form cannot be coded so that it can be printed back out unambiguously in Cyrillic, due to the non-agreement on transliteration. Roman letters, with or without diacritical marks, are keyed as Roman (regular) character set, whatever the language the letters themselves transliterate.

It is also proposed that the Cyrillic coding scheme include two single-character sets: one single-character Russian (East Slavic) Cyrillic set - including three pre-1918 characters - and one single-character (Other Slavic) Cyrillic set to handle the overflow, rather than to provide a double-character set for all Cyrillic characters.

Fig. 15 is a proposed code for Russian Cyrillic character set. It adopts as much as possible the characters common to many transliteration schemes, several of the more commonly used of which are shown in juxtaposition.

E. LOGICAL SIMILARITY OF BIBLIOGRAPHIC RECORDS

1. General Remarks. Groups of records in retrieval processes may be defined according to similarity of subject content, or in one of several other senses. One condition for search execution (the stage which performs the actual retrieval and assembly of the records or elements a user wishes to inspect) is the definition of rules to aid him in deciding what will be acceptable to him in terms of closeness of match of the file responses to his request, as he carries on his dialog. However, a user must formulate a request based on the clues available to him when he initiates his probe of the file, before the individual records or groups of records which satisfy these clues can be presented to him. Therefore, the most should be made of these clues in order to make searching in an on-line environment satisfactory.

2. Automatic Error Control in User-File Interaction. One characteristic of these clues is that they will often have error in them and thus an exact match on the request is not always the

"best" retrieval. "Near misses" or assistance in correcting the error should be provided. Accordingly, a task was defined to develop automatic techniques for error detection and correction of request messages at the terminal. Because of the linguistic and statistical nature of the task, a consultant with experience in this area was employed to start the work. A technique which we denote as "name compression by equivalence class algorithm" has been developed. The background and reasoning for the algorithm itself is described in a separate paper by James Dolby, included as Appendix I to this report. The purposes and provisional results of tests of the algorithm are summarized in the remainder of this section. Although the specific intent of this algorithm is error control, it has usefulness in file search beyond that. For this reason a more general perspective on the concept of "closeness" is presented first.

3. Closeness in Name Searching. A measure of closeness can have several dimensions, e.g., specific subject similarity or associativity, likeness of one or more properties such as set inclusion (several books by one author), and a number of other relations. It is possible to apply such a measure to relations between records in the file, and between records and requests made of the file. And it is possible to apply such a measure to parts of logical records, e.g., author names. We can consider two major classes of file organization and search, and of the requests that each can handle. The first class is the general problem of match between requests and file responses which are in some sense close to the request. The other class is a special case of the first, in which closeness is defined as exact match of the response to the request. The exact match problem, as the simpler of the two, is the logical place to start the research.

At the time of search specification, the user has some initial clues which may or may not be well-formulated. These clues, such as author name, subject terms, etc., are used to make up a search request. The request will be expressed through search keys transmitted through a terminal device. If the clue is an author name, the user will key in all or some portion of the name. Two problems arise: no response may be obtained from the file corresponding to the key as initially expressed, either 1) because no record at all is present which identically matches the request as expressed, or 2) the key may contain errors (e.g., form of name at variance with the form in the machine file, simple misspellings, phonetic misunderstandings, name changes, etc.), thereby causing either no response or "false drops." If the search key was accurately known at the time of input but response is nil or minimal, then other search aids such as associational and relational techniques can be invoked (i.e., searching under relaxed conditions), and the search can proceed or terminate when some kind or amount of useful search output has been obtained. Alternatively, the user might proceed as if the search key were inaccurate and expand his field of search as follows.

4. "Noisy" Matches. If the exact spelling of the name used as search key is not accurate ("guess-match"), an error correction technique becomes immediately useful. The facility should be able to provide capability to be invoked automatically or at the option of the user. It might be executed automatically if the user thought he had input the correct spelling of the author's name, but received a negative response from the first search of the author index file, for example. That is, the system should not be allowed to "give up" just because the user missed on his first try. The search control program would, upon notice of a negative search, retrieve synonymous names, if the name input in the request was spelled closely enough to that of one or more names existing in the file according to some probabilistically determined threshold. (The precise nature of the "equivalence class" facility is yet to be determined. It will probably be tested at first by establishing a file of fixed-length coded classes. An alternative method would be to generate interpretively the names which are members of the class, at the time of input. Another combination might be to invoke the algorithm inside the machine but have a list of classes at the console as a user aid.)

The possibilities for the desired name would then be displayed for consideration by the user. This is a kind of grouping of parts of records (e.g., a cluster of similar surnames of different authors) one of which may turn out to be the one which the user is seeking. The names are regarded as related in that they have close spellings, a linguistic property exploited in a number of name compression systems.

In this sense the error control feature acts as a filter on the request, self-activating upon certain conditions, on demand at other times. Although searching via author keys may not be the most important file access point,* keyboarding of proper names is thought to be the area most vulnerable to input errors. It is anticipated that the equivalence class technique can be extended to non-name searches, e.g., words in title and subject index files. Error correction will be useful in any of the usual searching situations, i.e., for the person who cannot remember the spelling of an author name for a book he has seen before, or to handle requests based on bibliographic references which may contain spelling errors (the "bad citation" problem).

---

*Recent research reveals that for the type of catalog search for a book with which the searcher has had previous contact, only a little over 20% of a sample of users surveyed could remember author name clues. However, it was pointed out that of the searches undertaken using author name as an access point (or author + title), the cause of failure was lack of a method of manipulating incorrect author or title information in order to make it operative in catalog searching. See Vaughan, Delores K. "Effectiveness of Book-Memory Data for Conventional Catalog Retrieval." In: Chicago. University. Graduate Library School. Requirements Study for Future Catalogs; Progress Report No. 2. Chicago, Mar. 1968. (NSF Grant GN 432), p. 53.

5. Equivalence Class Algorithm.

a. General objectives. The handling of large bibliographic files presents two levels of error control, as pointed out by Dolby.* They are 1) error detection and correction during the input (file generation) cycle or as a result of feedback by users to correct the file; and 2) the system's reduction of the effect of the user's own errors, committed as part of his search. The use of equivalence classes is not restricted to error control. The general objectives of a compression scheme may be:

(1) for automatic cross-referencing among similar names, where there is not an error involved, but a file authority situation.

(2) to confirm or establish a "guess" match through "noise" i.e., misspelling.

(3) to help eliminate misspelled words from the new records updating the file.

(4) to save keystrokes both to speed up the request and reduce keying error.

(5) to achieve data compression *per se*, in storage (reduce disk space for an index file and for programming convenience in processing fixed length entries).

Our motivation in developing the equivalence class algorithm is primarily as a user aid in relation to (1), (2), and (4) above.

b. ILR objectives. Work to date in ILR has stressed the first of two areas of immediate concern: proper names as match elements in catalog searches, and regular vocabulary such as words in titles. The approach taken in the work on author names was to find a way of gathering together "like" names systematically so as to identify similar spellings (and possible misspellings) but without over-identifying the list of names in a class. That is, a balance must be found between errors of exclusion and errors of inclusion. By a "balance" is meant a minimal error of exclusion while at the same time achieving a code that will produce the minimal amount of identification that will match names in a given group of "close" name-forms. Such lists of families of close names are termed "equivalence classes", and the most familiar forms are found in the conventional telephone directory, e.g., for variant spellings of the name "SMITH".

The algorithm is intended to function both 1) as an error filter in on-line interrogation of a file, and 2) as a general aid in search elaboration and request reformulation. This tool will provide a capability for access to the file through a mechanism not now feasible in the limited, passive cross-reference structures in conventional card catalogs. Not only will the user be able to come in through routes not presently available, but he will be able to receive support from the machine in dynamically correcting his spelling errors when he is attempting to probe the file and converge on a particular datum, whether it be title of book, a name, or other information. On the other hand, he will be able to expand his search in ways not presently available, e.g., to track down names or records when the similarity among them is not explicitly recorded in conventional cataloging information.

---

*See Appendix I.

c. Development of a name compression scheme. A spelling equivalent abbreviation algorithm for personal names may be designed to produce variable-length or fixed-length canonical forms, i.e., class codes. Variable length coding has been extensively tested but is generally rejected for use in operating systems due to the added difficulty in programming in comparison to fixed length fields. Also, code compression is enhanced by fixed length codes where no interword storage space is required.

Since no definitive data was available yet on the problem of deciding exactly how long a fixed-length code should be, or on the nature of errors introduced by truncating variable-length classes to a given fixed length, the approach taken was to create two versions of the algorithm for surname equivalence classes: both a variable-length algorithm and a fixed-length algorithm.

First a "hand-drawn" prototype variable-length algorithm was written, based on study of the set of classes given in a local telephone directory. The rules for this algorithm are listed in Fig. 16. The algorithms used in other compression schemes were then synthesized and modified. The resulting algorithm was visually tested on the equivalence classes in the phone book.

d. Results of variable length algorithm on telephone directory. Visual analysis of the results of the variable-length algorithm suggested that reasonably accurate matches could be achieved without excessive over-identification, that is, inclusion of widely variant names in a class that should have been excluded from it.

A sample of the telephone directory names and the variable-length class into which they were compressed by the first version of the algorithm is shown in Fig. 17.

The variable-length algorithm achieved a score of below 5% under-identification error and over 77% preservation of distinct identifications when evaluated against the original phone book system. Specifically, the phone directory contained 451 equivalence classes. The initial version of the algorithm only split 22 (4.9%) of the 451 classes and preserved 349 (77.4%) out of the 451.

FIGURE 16:

A SPELLING EQUIVALENT ABBREVIATION ALGORITHM FOR PERSONAL NAMES

Dolby Version 1 - Variable Length

1. Transform: McG to Mk, Mag to Mk, Mac to Mk, Mc to Mk.
2. Working from the right, recursively delete the second letter from each of the following letter pairs: dt, ld, nd, nt, rc, rd, rt, sc, sk, st.
3. Transform: x to ks, ce to se, ci to si, cy to sy. Consonant -ch to consonant -sh. All other occurrences of c to k, z to s, wr to r, dg to g, qu to k, t to d, ph to f (after the first letter).
4. Delete all consonants other than l, n, and r, which precede the letter k (after the first letter).
5. Delete one letter from any doubled consonant.
6. Transform pf# to p#, #pf to #f, vowel -gh# to vowel -f#, consonant -gh# to consonant -g#, and delete all other occurrences of gh. (# is the word - beginning and word-ending marker.)
7. Replace the first vowel in the name by the symbol "*".
8. Delete all remaining vowels.
9. Delete all occurrences of w or h after the first letter in the word.

(NOTE: vowels are defined as a, e, i, o, u, y.)

FIGURE 17:

EQUIVALENCE CLASS COMPUTATION (MANUAL)

Version 1

A portion of a list of personal-name equivalence classes from the Palo Alto-Los Altos Telephone Directory, arranged according to the variable length compression code (with the vowel marker * treated as an A for ordering). (1)

Variable-length "Dolby Code"	Names Belonging to Class
*BL	Abel, Abele, Abell, Able
*BRMS	Abrahams, Abrams
*BRMSN	Abrahamson, Abramson
*D	Eddy, Eddie
*DMNS	Edmonds, Edmunds
*DMNSN	Edmondson, Edmundson
*DMS	Adams, Addems
*GN	Eagan, Egan, Eggen
*GR	Jaeger, Yaeger, Yeager (2)
*KN	Aiken, Aikin, Aitken
*KNS	Adkins, Akins
*KR	Acker, Aker
*KR	Eckard, Eckardt, Eckart, Eckert, Eckhardt
*KS	Oakes, Oaks, Ochs
*LBRD	Albright, Allbright
*LD	Elliot, Elliott
*LN	Allan, Allen, Allyn
*LSN	Ohlsen, Olesen, Olsen, Olson, Olsson
*LVR	Oliveira, Olivera, Olivero
*MS	Ames, Eames
*NGL	Engel, Engle, Ingle
*NL	O'Neal, O'Neil, O'Neill
*NRS	Andrews, Andrus
*NRSN	Andersen, Anderson, Andreasen
*NS	Ennis, Enos
*RKSN	Erichsen, Erickson, Ericson, Ericsson, Eriksen

Notes: (1) A small number of directory entries that do not bear on the immediate problem have been deleted from the list: Bell's see also Bells: Co-op see also Co-operative; Palo Alto Clinic see also Palo Alto Medical Clinic; St. see also Saint; etc.

(2) Names whose compressed codes do not match the one given in the first column (and hence represent weaknesses in the algorithm and/or the directory groupings) are underlined.

An analysis of the under-identification errors (i.e., close names that should have been included in a given class or classes that should have been split, according to the phone book) is in the Dolby paper in Appendix I.

e. Results of testing on catalog data. A program was then written to implement the variable-length algorithm by computer. This version was tested on a sample of 50,000 names in the Santa Cruz machine-form author file. Fig. 18 shows some selected names and the "canonical forms" which were computed for families of close names.

It was then decided to construct a second, fixed-length version of the algorithm to discern the kinds of errors that might be introduced by truncation of the variable-length classes to some standard length. The initial list of rules for Version 2 is presented in Fig. 19. This version has not yet been implemented, but computer testing of it will be carried out in the next phase of the project. The fixed-length version incorporates modifications suggested by analysis of results of the program for the variable-length classes together with separate "hand" analysis of the effect of truncation. We intend to program the improved fixed-length algorithm and evaluate it comparatively with other equivalence class schemes such as SOUNDEX.*

f. Effect of truncation to create fixed-length classes. Initial analysis of the variable-length algorithm revealed that simple truncation will not generate errors of under-identification but will lead to further over-identification, i.e., inclusion of disparate names in a class, that should have been excluded. The simple truncation to the left-most seven characters of the classes produced from the phone book names, introduced no losses from combining too many classes (over-identification). However, reduction to a code length of four causes a jump in the cumulative over-identification losses (i.e., too many names being included in a given class that should not be included due to wide variations in spelling). An optimal length of five characters for a straightforwardly-truncated class code appears to be attainable, however, in advance of any testing on a large file. Since it is desirable to obtain the shortest possible class, a method other than simple truncation was sought. A possible solution is further refinement of the algorithm itself, ending up with a final fixed length of four. The procedure proposed acts via selective removal of some of the remaining characters, such as vowel-marker deletion from the longer words and insertion of additional vowel-markers in the very short words. Preliminary analysis indicates that an improvement is achieved. Manual application of the version 2 algorithm on the phone book classes produced fixed-length compression codes

---

*Becker, Joseph and Robert M. Hayes. Information Storage and Retrieval: Tools, Elements, Theories. New York, Wiley, 1963. pp. 143-144.

FIGURE 18:
EQUIVALENCE CLASS COMPUTATION (COMPUTER)

Version 1

Variable-length "Dolby Code"	Names Belonging to Class (1)
*BRN	O'Brian, O'Brien
*NL	O'Neal, O'Neil, O'Neill
*BD	Abbatt, Abbot, Abbott, Abetti, Ebbit, Obieta
*BR	Aubert, Auboyer, Aubrey, Aubry, Ibert, Ybarra
*BRM	Abraham, Ibrahim
*BRMS	Abrahams, Abrams
*BRMSN	Abrahamsen, Abramson
*DRS	Edwardes, Edwards, Idriess
*LPR	Allport, Alpers, Alpert
*LS	Ellis, Alas, Eales, Eells, Elias, Ellis, Else, Elst, Elwes, Olds, Olesha
*LSN	Allison, Alston, Ellison, Elsen, Elson, Elston, Ohlsen, Oleson, Olsen, Olson
*MR	Amery, Amory, Aymar, Aymard, Emery, Immer
*MRN	Amerine, Amrine, Emerson, Emerton, Emmerson
*NG	Ewing, Ienaga, Inge, Iongh, Ong, Yanaga, Yang, Youge, Young, Younge
*NN	Anand, Annan, Anthony, Antin, Antoni, Ennin, Onnen, Unwin, Yenawine, Yohannan
*R	Airey, Ard, Arey, Arrow, Auer, Aury, Ayer, Ayers, Ayre, Eayrs, Ewers, Ewert, Eyre, Ihara, Irie, Iyer, Ore, Orr, Orrey, Orth
*SBRN	Ashburn, Ashburne, Osborn, Osborne, Osbourne
B*K	Bach, Back, Baikie, Bakke, Beach, Beachey, Beck, Becke, Beke, Bewick, Biek, Boak, Bocca, Bock, Bodky, Boeck, Boeke, Bok, Buck

---

(1) Selected from Santa Cruz machine author file.

FIGURE 19:
ABBREVIATION ALGORITHM FOR PERSONAL NAMES

Version 2 - Fixed Length

(This version incorporates refinements proposed as a result of applying Version 1 of the algorithm to about 50,000 author names selected from the UC Santa Cruz machine catalog file. These rules should be regarded as interim, in that further revisions may be suggested after additional analysis.)

For each sur-name:

1. Remove all blanks, hyphens, and apostrophes.
2. Transform McG, Mag, Mac or Mc appearing at the beginning of a name, to Mk.
3. Working from the right, recursively delete the second letter from each of the following letter pairs:

   dt, ld, nd, nt, rc, rd, rt, sc, sk, st
4. Transform: x to ks, ce to se, ci to si, cy to sy; consonant-ch to consonant-sh, all other occurrences of c to k, z to s, wr to r, dg to g, qu to k, t to d, ph to f.
5. Delete all consonants other than l, n, and r which precede the letter k (after the first letter).
6. Delete a final e if it occurs.
7. Delete one letter from any doubled letter.
8. Transform pf# to p#, #pf to #f, vowel-gh# to vowel-f#, consonant-gh# to consonant-g#, and delete all other occurrences of gh. (# is the word-beginning and word-ending marker).
9. Transform v to f.
10. Replace each of the first two vowel strings by "*". Here we consider a vowel to be any of the characters A,E,I,O,U,Y.
11. Delete all remaining vowels.
12. Delete all occurrences of w or h after the first letter.
13. If the name is longer than 4 characters, drop one final s.
14. Truncate to six characters from right end.
15. Reduce name to 4 characters by first removing the rightmost * if length$\geq$5. If still not reduced to 4, remove second * (can't be more than two *) and then truncate if need be. If resultant length is

FIGURE 19 (Cont.):

ABBREVIATION ALGORITHM FOR PERSONAL NAMES

Version 2 - Fixed Length

now less than or equal to 4 letters, retain one *, don't remove the *.

16. If the name is less than 4 characters, pad with blanks at the right, to get a uniform length of 4 characters total.

of length four, resulting in 361 distinct classes or 80% of the 451 original classes in the directory. Improper splits (failure of the algorithm to identify a name with its proper class) occurred in 24 or 5.3% of the classes.

6. Comparison with Present Manual Catalogs. It may still be wondered why the equivalence class idea is advantageous. The conventional card catalog provides a rather rigidly constructed "syndetic" apparatus to guide the user to his "target" - reference structures that lead, for example, from forms of name requested to the specific form of name used in recording particular file items, and from specific known names to a list of pertinent related names (e.g., to a pseudonym: from "Dannay, Frederic see entries under Queen, Ellery; Ross, Barnaby;" etc.)

The subject cross referencing system is even more highly constrained - the dictionary of synonyms and other link terms is dispersed throughout the catalog; only the user who has access to the "authority list" can even try to assemble systematically all the possible terms under which he might wish to search. Instances of related terms are often explicitly listed as "see also's", but the distance separating any two similar terms or names may be great, and thus printing out terms on either side of the requested one is frequently unproductive. Some method must be employed that overcomes the linear array of the file.

The manual file, e.g., the telephone book or card catalog, usually makes limited, rudimentary efforts to help the user correct his own errors. "See" and "see also" references are inserted for variant spellings of names in the same and different languages. These are usually permuted general class references, e.g.,

ACCESS 1: SMITH	ACCESS 2: SMYTH	ACCESS 3: SMYTHE	ACCESS 4: SCHMITH
see also	see also	see also	see also
SMYTH	SMITH	SMITH	SMITH
SMYTHE	SMYTHE	SMYTH	SMYTH
SCHMITH	SCHMITH	SCHMITH	SMYTHE

and

CATHERINE

For sovereigns, princesses of sovereign houses, and saints:

Bohemian:	see	Katerina
Dutch:	"	Katherina
English:	"	Catherine
. . .		. . .

This apparatus not only is cumbersome to use but complicated to input and maintain. It also introduces a degree of redundancy into the file which can be precluded by mechanisms such as equivalence classes.

7. Example of Use of Equivalence Class in On-line Mode. It is useful to run through a hypothetical exercise to see how the name compacting algorithm might be of service in an on-line mode.

The user desires to check a name in the file to see whether the name exists, and if it does, what relation it bears to the file content, e.g., as author of a book, as subject of a biography, etc. He is assumed not to have a "verified" version of the spelling of the name in mind, i.e., no knowledge of the exact spelling of the name he wants as it exists in the file and even if he did, it would not necessarily correspond exactly to the particular form in which the name and its associated elements are established as a catalog heading.

A necessary stage in the protocol of interrogating the file is to decide what to do as the next step. Let us assume that the first action is to key in the full or partial surname. (In our first experiments this will be the first n characters of the surname, depending on the kind of rules that are constructed from an analysis of uniqueness of identification mentioned in Section II. In later experiments, it would be interesting to test whether a relatively free-form mnemonic* could be employed, so that the user could input n characters according to general rules and have the computer perform the match with the equivalence class mechanism rather than the user selecting an assigned code from a list.)

a. For example, if the surname key is used and the user thinks the name is GARNETT, A.C., he would key in "GARN". Assume that a degree of match is found at this level of identification. The computer could then display a sequence of messages describing the file contents, in response: e.g.,

> "There are 14 books by authors named Garn, Garner, and Garnett, in the file you have queried.
>
> INDICATE WHICH AUTHOR YOU WISH TO DISPLAY
>
> ·GARN ·GARNER ·GARNETT"

If the user requests a display sequence for "Garnett", the following message might be output:

> "Option 1: There are 7 books by authors named Garnett in the file.

---

*Jackson, Michael. "Mnemonics." Datamation, v. 13 (Apr. 1967), pp. 26-28.

Option 2: There are 5 books by authors named Garnett with A. for one of their initials in the file.

Option 3: There are 3 titles by authors named Garnett, A.C., in the file.

Option 4: There are 2 titles by author whose name is Garnett, Arthur C., in the file.

Option 5: There is 1 title by the author whose name is Garnett, Arthur Campbell, in the file.

Please select option you wish to explore further."

Assume the user goes on to select "Garnett, Arthur C." as the requested author. In this case he need only further select from two titles.

By such a process of sequential query of the file contents, the user is able gradually to converge on the name or document he desires, or confirm that it does not exist, assuming he can make these decisions from the file information alone. Of course, in large catalogs it may be necessary to employ gradually (in an iterative fashion) more and more information about the name or document(s) sought, such as titles of honor, birth/death dates, publication date, and other distinguishing data. Such data would only be displayed when needed, to permit the user to make distinctions among dense clusters of identical names which are identified positively only by such subsidiary elements.*

The actual machine match at each stage would be made on the portion of the name required to respond to the user's specification. He may wish to expand or contract his notion of matching. Should he decide to exercise the option which would lead to a display of a run of very close names, for example, the action of the algorithm controlling this would be based on density factors of the actual file (number of identical names which are only distinguished by affixed information, the number of titles per unique author name, etc.). If there are hundreds of entries under a run of similar surnames (e.g., Allen) the search control could step through presentations of various levels of summary information.

b. The other half of the author searching procedure is concerned with expanding the field of search when no match is encountered. Let us assume that the surname match has failed and hence we are not interested in the initial spelling but rather wish to interrogate the file for similar surnames. For example, assume a search on the class of names beginning with "Aarons". If the user did not know which spelling to guess at he could reverse the process and key in the equivalence class code "*RNS" from a list at the terminal, or if he did wish to guess at a name, he could key either a full surname or an abbreviated key for it, e.g., ARON. The response in the case of the full surname might be "no match" but that there are other spellings. The dialog would then proceed as above. Our research so far has served to point up the complexity of even the simplest cases.

---

*This procedure for search purposes is analogous to the "no conflict" policy employed in cataloging, where no identifying information need be affixed to name headings except where needed to distinguish between two different persons with identical names.

IV. DATA BASE DEVELOPMENT

A. GENERAL

1. Recent Data Base Projects. There has been much recent activity in the development of machine-form bibliographic files. For example, the Library of Congress provided a limited amount of English language cataloging in machine-form to a group of participating libraries through its MAchine Readable Catalog (MARC) Pilot Project. Harvard University has a closely related project to provide non-LC bibliographic records in the MARC format. In addition to these directly related projects, a number of universities have begun developing machine data bases of bibliographic records, among which are the Santa Cruz campus of the University of California, Stanford, MIT, the University of Chicago, and the University of Toronto.

In early 1967 the Institute of Library Research completed an intensive investigation of book-form catalogs for the California State Library.* During the study, a format convertible to the LC MARC I format was developed and used in an analysis of the effort required to obtain a machine-form data base from bibliographic records. It was from this basis that we began our investigations of data base development.

2. Current Effort. In developing the data base for the facility, bibliographic records for both monographs and journal articles are being incorporated in the system as the initial forms of library materials to be represented by machine records. Our first effort was devoted to establishing a set of procedures to obtain the initial monograph data base for the study. Under the funding granted, it was anticipated that a data base containing at least 200,000 titles in the Roman alphabet could be developed. To obtain a file of this size, it was recognized that some original input would be necessary. In addition, we decided to make an extensive study of the procedures by which existing data bases could be integrated into our system along with our original input. From this study, we devised a method for extracting the relevant information from some of these sources in order to reduce original input as much as possible. The incorporation of these materials requires computer programs which we are specifying and coding in order to translate the existing machine data bases.

---

*Cartwright, K.L. and R.M. Shoffner. Catalogs in Book Form: A Research Study of their Implications for the California State Library and the California Union Catalog, with a Design for their Implementaion. Berkeley Institute of Library Research, University of California, 1967.

Where original input is to be accomplished in the development of the data base, the source of this material will be the recent cataloging output of the nine campuses of the University. In order to minimize delay in the establishment of our data base, we started up the process for original input in parallel with this programming. We took care to ensure a minimum likelihood of redundant conversion of the various machine-form information. It was known that the Harvard project is concentrating on materials of 1967 and subsequent publication date. The MARC records were begun in mid-1966 and do not include retrospective materials. As a result, virtually no materials prior to 1966 publication date are included in the MARC I file. The materials of the Stanford University collection and the University of California Santa Cruz collection are both primarily undergraduate collections. The remaining materials available tend to be concentrated in specific subject areas.

As a result, we anticipate that the majority of machine records for materials of publication date later than 1967 will be fully covered by the combination of the Library of Congress and Harvard University projects. Therefore no original encoding of materials for this publication region has been undertaken. With respect to some of the materials of 1966 and later publication, it is our expectation that at some point, the records now existing in LC MARC I files will be supplied to users in MARC II format by the Library of Congress. In anticipation of having both these upgraded MARC I records and current MARC II records from LC, materials with a publication date prior to 1966 only, are being used for original encoding in our project.

Overlap with the undergraduate collections previously mentioned can be avoided in two ways: first, by selecting material which appears to be relevant to a research library collection but not to a basic undergraduate collection; and second, by checking questionable materials against the printed catalogs and lists of these collections which are available.

In our input procedure, the encoding of the bibliographic data to identify the logical content of the record and the keying to provide the record in machine form are performed as two separate processes. The encoding process requires training in library cataloging in order to identify the logical content of the bibliographic record. The keying process does not require such training. Rather, the central issues in this process are the speed and accuracy with which it can be performed. This performance is enhanced given that virtually all decisions have been made about what is to be keyed.

By separating the keying operation and by making all information required by the computer input program explicit in the data stream, the keying can be performed on any device such as keypunch, paper tape typewriter, on-line typewriter, or standard typewriter with optical scanning equipment. This means, for example, that the computer program is organized to function without regard to the end of a tab card in a keypunch operation.

The separation of these processes is particularly useful since there exist commercial firms specializing in the keying aspects of data conversion, to which this task can be contracted. By separating this part from the other procedures which require knowledge of bibliographic records, we can utilize these services with most efficiency.

An important aspect of any production procedure is the control that it provides for maintaining the appropriate cost and quality of the product provided. We have been studying the trade-off of cost and quality and we have established a statistical quality control procedure to maintain adequate accuracy in the conversion process without excessive expenditure for this control.

B. STRATEGIES OF CONVERSION

This section is addressed to two closely related conversion strategies which are often mentioned but which have not received careful consideration. They are the use of existing machine data bases and joint efforts in which the conversion is shared among several libraries. A critical aspect of both of these strategies is the amount of overlap that exists between the catalog data bases. In the material that follows we consider this overlap as a variable and show the impact that it has on the effectiveness of these strategies.

1. Utilization of an Existing Data Base. If a data base already exists in machine form, it is possible to search it in order to extract from it matching records to be inserted into the new data base being assembled. There are three elements which are important to the unit cost of the records obtained by this approach: the cost of the search; the expected number of records that will match and thus be useful; and the translation costs to convert the record to the format of the new data base.

The search cost is dependent upon the kind of arrangement or degree of order of the two files - the old and the new, the size of the files, and the match method used. In general, the least expensive method is to have the two files sorted to the same order and then merge them. Although sorting time is not a strictly linear function of the number of records, only a small error is introduced by considering it linear. Used for the purpose of a search, merging time is approximately a linear function of the number of records in the larger of the two files by this approach. The search cost can be characterized as a sort cost per record times the number of records in the new file (assuming its initial order to be different from that of the new data base), plus a merge cost per record times the number of records in the larger of the two files. The translation cost depends upon the degree to which the format and content of the existing and the new records are the same. In any event, the translation cost is a linear function of the number of records translated. That number can be characterized as a percent of the the total number of records to be included in the new file.

We now develop an algebraic expression for the cost per converted record. Let the number of records in the new data base be represented as "V_{new}". Assuming that it is the larger of the two data bases (true for our case) the search cost is:

$$\$\text{search} = (\$\text{sort} + \$\text{merge})\ V_{new} \qquad (1)$$

where "\$sort" is the sort cost per record and "\$merge" is the search-by-merge cost per record (batch mode).

From this, we obtain as the next expression:

$$\begin{aligned} \$_{conv} &= (\$\text{search} + \$\text{trans} \cdot R_{hit} \cdot V_{new})/(R_{hit} \cdot V_{new}) \qquad (2) \\ &= \$_{search}/(R_{hit} \cdot V_{new}) + \$_{trans} \\ &= (\$\text{sort} + \$\text{merge})/R_{hit} + \$\text{trans} \end{aligned}$$

where

"\$trans" is the translation cost per record;

"R_{hit}" is the ratio of the records V in the new file which are found in the existing data base; and

"\$conv" is the cost per record of obtaining the new data base by searching, extracting, and translating records from the old data base.

If our purpose is to choose the approach which requires the least cost to obtain the new data base, we may compare this approach to that of a straightforward conversion of the new data base, which makes no use of the existing data base. As before, we can consider the cost as a linear function of the size of the new file and express the per record input cost as "\$new". We can now set up an inequality such that if the expression is true, we could choose to use the existing data base in the development of the new one:

$$\begin{aligned} \$_{new} &\geq \$_{conv} \\ &\geq (\$_{sort} + \$_{merge})/R_{hit} + \$_{trans} \qquad (3) \end{aligned}$$

From this, it is clear that in considering these alternatives an immediate check can be made to confirm that the cost of translating an existing record is indeed less than that of the original conversion of the desired record. Variations in format or content could be great enough in themselves to reverse the direction of the inequality. Assuming, however, that this is not the case, we can proceed to rearrange the terms (since all terms are positive, the direction of the inequality will be preserved):

$$R_{hit} \geqq (\$_{sort} + \$_{merge})/(\$_{new} - \$_{trans}) \qquad (4)$$

This expression simply says that for search to be justifiable, the ratio of hits must be greater than/equal to the ratio of the search cost (sort-plus-merge) to the difference between the straightforward conversion cost (non-translated file) and the conversion cost by the method of translation of an existing machine file.

Another approach to the use of the existing data base is to allow the files to remain in their original (perhaps unsorted) order and to make a random search for each desired record. For example, such a search could be performed over a terminal to an on-line file. In this case, our general analysis is the same except that we have a search cost (on-line mode), expressed as a linear function of the number of records, which replaces the sorting and merging cost. This can be substituted in expression (4) as "$srch":

$$R_{hit} \geqq \$_{srch}/(\$_{new} - \$_{trans}) \qquad (5)$$

Thus, the analysis is applicable to either situation. Further, for on-line search to be less expensive than the sorting and merging approach, the following must hold:

$$\$_{srch} < (\$_{sort} + \$_{merge}). \qquad (6)$$

2. Joint Conversion Efforts. As mentioned earlier, it is often the case that a joint conversion effort is proposed as a desirable strategy. Given that all the records from a group of separate source files are to be converted, there are still many questions that must be answered in order to determine how to go about the conversion. First, is there enough overlap between the files to make it worthwhile to try to avoid multiple conversion of the "same" record? This is the question we were addressing in the previous section, in which we formulated an equation which can be used to compute the minimum amount of overlap needed to justify a procedure to prevent duplicate conversion. If the overlap is sufficient, which records should be converted first? Can some files be converted independently?

Can some files be converted in parallel, or is there a single desirable conversion sequence?

Careful analysis shows that this is a less complicated series of questions than it first appears. First, the total amount of direct conversion effort will be the same regardless of the sequence in which it is performed. Therefore, the factor which affects the total conversion cost is the search for those records already converted to machine form. Therefore, the proper conversion strategy should minimize the total search cost, incurred during the course of converting the whole "new" file, either by eliminating the search altogether, by reducing the number of searches, by reducing the search cost per record, or by a combination of these.

Some approaches which eliminate or reduce the number of searches will do so at the risk of a certain amount of unintended duplication of records converted. We will first consider only approaches that do not risk duplicate conversion.

a. Conversion with guaranteed control of duplication. The major source of search cost is search time, whether in a computer or a manual system. Let us use the following notation:

V_i = the number of records in file i;

R_i = the ratio of the total number of records in file i, which are in some other file as well;

$R_{i,j}$ = the ratio of the total number of records in file i, which are also in file j;

$\$_{isrch}$ = the cost per record of searching another file for duplicates while converting file i.

The total search cost is thus:

$$\$_{srch} = \sum_{i=2}^{n} V_i \$_{isrch} \qquad (7)$$

Note that the summation index begins at 2 because there is no need to perform a search when the first file is being converted, since no machine file exists yet.

The search cost per record is dependent upon the file size, that is, upon the number of records already converted, over which the search for duplicates takes place. Although there are many different ways in which that search can be performed, we assume it will be one of a binary nature in which the search time is proportional to the logarithm of the file size. Thus,

$$\$_{isrch} = k \log_2 \left(\sum_{j=1}^{i-1} V_j - \sum_{j=1}^{i-2} \sum_{h=j+1}^{i-1} R_{j,h} V_j \right) \qquad (8)$$

This formulation of search cost is useful primarily for random search.

Determining the sequence in which a group of source files are to be converted has two parts: 1) the general case of selecting the next file to be converted; and 2) the special case of selecting the first file.

The rule for the general case is straightforward. The file to be converted next is the one which has the lowest ratio of records which are not yet in the converted file, to total records in this file. This rule balances out the execution of the maximum possible searches of the converted file at its current size (as would be provided by selecting the largest remaining file for conversion next), against the addition to the file of the least number of records possible (as would be provided by selecting the remaining file with the fewest records not already converted).

A batch oriented search is formulated in a manner similar to (8), but it is linear:

$$\$_{imerge} = g \left(\sum_{j=1}^{i} V_j - \sum_{j=1}^{i-2} \sum_{h=j+1}^{i-1} R_{j,h} V_j \right) \qquad (9)$$

where "$imerge" is the total cost for the search of all of the records in the new file i.

The use of this search method changes the rule for selecting the least cost conversion sequence. Because it is a batch process, the cost is not sensitive to the total records in the file to be selected. Therefore, the appropriate rule is to select for conversion next that remaining file which has the fewest records that have not yet been converted (i.e., the most overlap with the other files). This has the effect of keeping the growth in total size of the converted file at smaller increments and thus the merges as small as possible.

Now let us consider the special case of selection of the first file which will be converted without pre-sorting and without search since there is no data base already in machine form. In the case of a random search it will be the largest file for which the overall duplication rate with the other files, R_i, satisfies the expression defined for R_{hit}:

$$R_{hit} \geq \$_{srch}/(\$_{new} - \$_{modif}) \tag{10}$$

"$modif" in the equation refers to either the cost of converting records from manual files or the cost of translating from an existing data base, when search is used.

By this rule, the most searches will be avoided for a file which would have been converted with search at some point in the sequence. This search avoidance reduces the cost more than converting the large file increases it.

For a batch processing procedure, the first file is selected according to the general rule, i.e., the smallest file first. The reasoning for this was given above.

b. Independent/parallel file conversions. To this point we have discussed only the general characteristics of the files to be converted. On this level of analysis it would not be possible to define a method by which independent conversion of several files in parallel could take place. The key issue in defining such a method is to be able to predict on the basis of some set of readily observable characteristics of the record to be converted, whether or not it is unique to this file. If this prediction can be made to a high enough level of accuracy, then ipso facto it would not be worth it to perform searches for these probable unique records. The duplication rate needed to make search and modification (integration of duplicates with variant elements such as call number) less expensive than direct conversion without search was shown to be:

$$R_{hit} \geq (\$_{sort} + \$_{merge})/(\$_{new} - \$_{modif}) \tag{11}$$

Recall that search is not economic if the hit ratio is less than the ratio of search cost/conversion cost differential. So long as the error of estimate is not so great that the actual hit rate satisfies this expression while the estimated hit rate does not satisfy it, then these records having low expectation of duplication can more economically be converted independently, without search.

By this procedure, then, the records in all the files, which are predicted to be unique could be separated and converted in any order whatsoever.

Parallel conversion could be accommodated by the strategy of segmenting the files into sub-groups, such as letter groups by main entry or title, and allocating different sub-groups for conversion at a given time. This technique, however, implies later search and reconciliation of the file segments.

As a result, it does not attempt to minimize the total conversion effort as do the approaches we have presented here.

3. An Example: The Catalog Supplement. To develop the data base for our facility by original conversion, we are using the catalog records of the University's Catalog Supplement. These are records of material which were new cataloging (not necessarily new publications) to the campuses during the period 1963-67. We have made extensive studies of these records and are thus in a position to provide examples of the use of the expressions defined in the earlier parts of this section. This discussion, it should be noted, is intended only as an illustration, however, since many of the estimates are not verified. Assume the parameters have the following values:

$\$_{sort}$ = \$0.05 per record

$\$_{merge}$ = 0.01 " "

$\$_{new}$ = 0.75 " "

$\$_{modif}$ = 0.25 " "

$\$_{srch}$ = 0.15 " "

For a manual approach to the search, R_{hit} must satisfy the following relation:

$$R_{hit} \geqslant (0.05 + 0.01)/(0.75 - 0.25), \tag{12}$$

i.e.,

$$\geqslant 0.12 \tag{13}$$

For an on-line search approach, it must satisfy

$$R_{hit} \geqslant (0.15)/(0.75 - 0.25) \tag{14}$$

i.e.,

$$\geqslant 0.30 \tag{15}$$

Total estimated file sizes, by campus, are as follows:

Berkeley	214,000
Davis	131,300
Irvine	53,500
Los Angeles	276,200
Riverside	99,500
San Diego	128,700
San Francisco	17,500
Santa Barbara	126,100
Santa Cruz*	-
Total -	1,046,800

At this time we do not have estimates of the duplication rates among these files. However, we believe that it can be reasonably hypothesized that duplication rate and file size are negatively corrrelated. That is, the larger the file, the less the duplication rate.

Using this hypothesis and the selection rules defined previously, we would first convert the largest of the files, that from Berkeley. The next file would be the one estimated to have the greatest overlap among the nine. It would be searched against the machine-sorted Berkeley author file, after which we would convert the unique titles and input added locations for the matches. The next highest overlap file would follow, and so on down to the last remaining file with the greatest residual uniqueness.

In this situation we would not take San Francisco as the file with the highest duplication rate, even though it is the smallest file. Because it represents a special collection (to support a medical school) our hypothesis of size and duplication rate is not likely to hold. Therefore, we would convert it next to last. All other campus files would be chosen in ascending order of file size.

There are two characteristics which we believe will be useful for predicting duplication rate. If so, they will bear on the search/convert decision. They are the language of the record and the indicated date of publication of the material. Our hypothesis is that duplication rate is highest for English records, next highest for records in other Roman languages, and lowest for non-Roman languages. Also, the older the material, the less the duplication rate. If the record was seen to have a particular language and a publication date that falls before a certain date, the card would be converted without search of the machine file, in the expectation that duplication would be very low. Otherwise, it would be searched against the data base.

---

*The Santa Cruz file is now in machine-form and was not estimated at this time.

In order to catch unintended duplication, a check could be made automatically as part of the file maintenance process (or manually, off-line) to compare possible matches to see if they are truly duplicates. If found to be matches, post-conversion modifying would then be done.

To check these hypotheses, we took a sample of approximately 7000 records and counted the duplication rate as a function of language and publication date. The results are given in Fig. 20. It should be noted that this is an overall duplication rate: it is computed as a ratio to the resulting file after duplicates are removed, rather than to the inclusive source file with duplicates intact. The results for English conform very well to our hypothesis. It would appear that 1962 would be a reasonable cut-off date for the use of on-line search, on an experimental basis.

No consistent pattern appears for publication date for the non-English materials. Given the outcome of the work with English, more study of these materials may be needed. At any rate, it appears that for English language materials there is a high likelihood that if we used 1962 as the cut-off date for searching, not to search items with a date prior to 1962 would give around a 30% chance of converting an item twice. Whether this is acceptable will depend upon verification of the various estimates which were used in the computation of R_{hit}.

C. TRANSLATION OF EXISTING MACHINE FILES

1. General. In order to investigate the automatic translation to our format of a file in another format for data base development, we obtained copies of the magnetic tapes of the catalog of the Library of the University of California Santa Cruz campus.

The reasons for the decision to use the Santa Cruz file rather than the LC MARC I file, arose from our analysis of the UC catalog file.

First, the use of Library of Congress MARC I tapes for extraction was likely to be useful for less than 10% of the records in our source file. This is the case because the MARC I file contains English language materials, the preponderance of which were published since mid-1966. At present, probably about half of such material which has been published has been included in the MARC I file. Therefore, we decided to concentrate on the development of translation procedures for the Santa Cruz tapes since: 1) they contain approximately 90,000 catalog entries and 2) the Santa Cruz file should have considerable duplication with at least two of the other campuses.

FIGURE 20:
DISTRIBUTION OF DUPLICATE TITLES AS A FUNCTION OF PUBLICATION DATE

A. TITLES IN THE ENGLISH LANGUAGE

No. of Copies	Year of Publication	1967	1966	1965	1964	1963	1962	1955 to 1961	1950 to 1954	Prior to 1950	no date
1	% of year	.83	.53	.49	.20	.56	.67	.82	.81	.86	100
2	% of year	.17	.18	.19	.10	.14	.17	.12	.18	.10	
3	% of year		.10	.10	.04	.14	.10	.06		.02	
4	% of year		.10	.06	.60	.03	.05			.02	
5	% of year		.03	.07	.03	.08					
6	% of year		.02	.07	.03	.04					
7	% of year		.03	.02							
8	% of year										
9	% of year		.01								
Total %		100	100	100	100	99	99	100	99	100	100

% of year: This is the percent of titles (in the language indicated) in the sample that were published in the given year, having the given number of copies.

B. TITLES IN LANGUAGES OTHER THAN ENGLISH THAT USE A ROMAN ALPHABET

No. of Copies	Year of Publication	1967	1966	1965	1964	1963	1962	1955 to 1961	1950 to 1954	prior to 1950	no date
1	% of yr		.83	.79	.78	.83	.79	.83	.79	.92	100
2	% of yr		.12	.16	.14	.11	.13	.15	.14	.07	
3	% of yr		.03	.03	.03	.03	.03	.03	.06	.01	
4	% of yr		.02	.02	.03	.03	.05				
5	% of yr				.02	.005			.01		
6	% of yr										
7	% of yr										
8	% of yr										
9	% of yr										
Total %			100	100	100	100	100	101	100	100	100

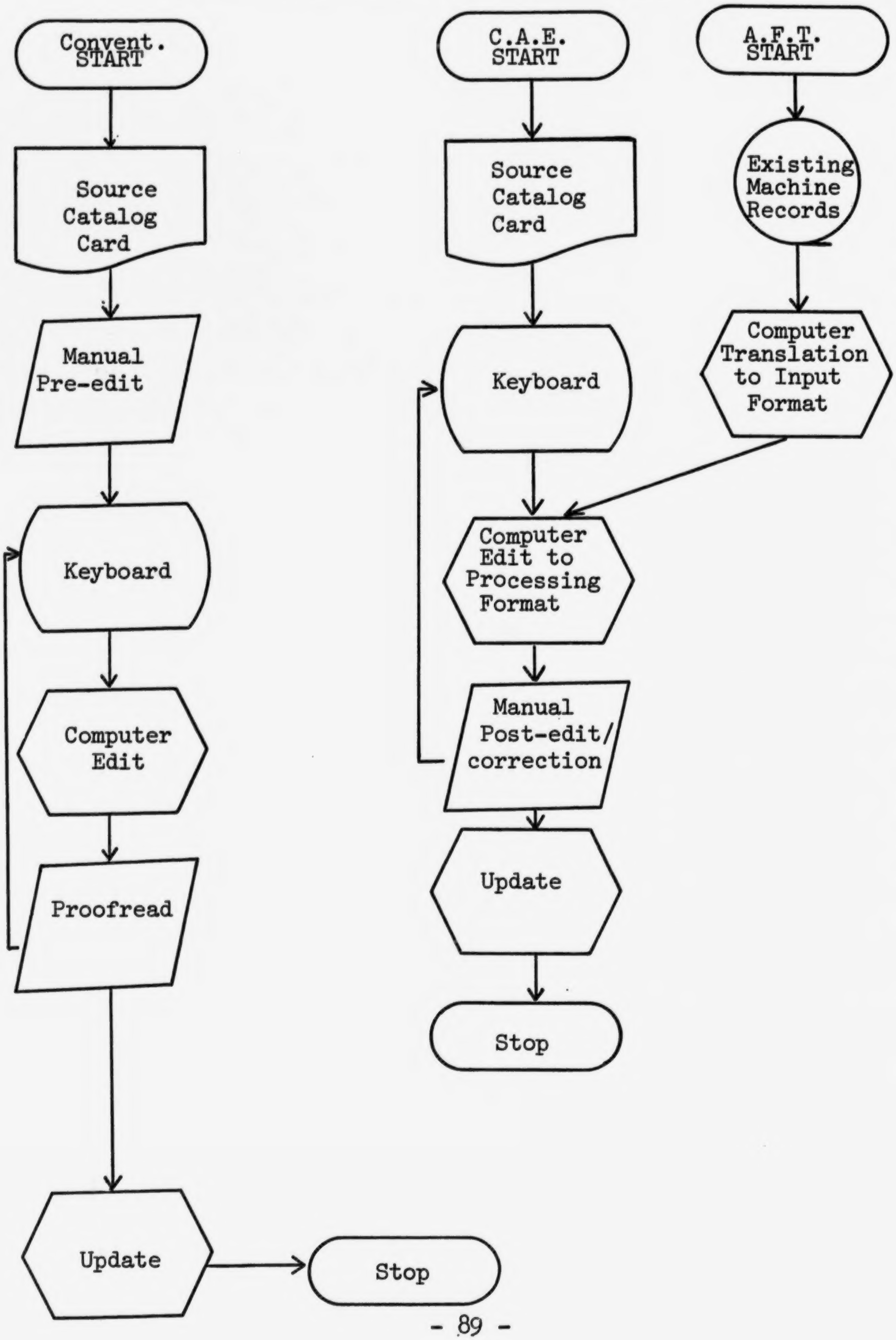
C. TITLES IN LANGUAGES THAT USE A NON-ROMAN ALPHABET INCLUDES TRANSLITERATED TITLES

No. of Copies	Year of Publication	1967	1966	1965	1964	1963	1962	1955 to 1961	1950 to 1954	prior to 1950	no date
1	% of yr		.91	.93	.98	.95	.96	100	100	100	100
2	% of yr		.09	.07	.02	.05	.04				
Total %			100	100	100	100	100	100	100	100	100

The automatic translation of records is of considerable importance, not only for the conversion of the Santa Cruz materials as such, but because it represents a special solution to the general problem of converting from a less specific format (such as the Library of Congress MARC I format or the Stanford University Undergraduate Catalog format) to a more specific and complex format (such as the Library of Congress MARC II format). To the extent that automatic conversion is successful, materials which have been converted in less specific formats will be more immediately useful in systems based on the MARC II format. Also, the continuing process of input in the MARC II format requires that many of the encoding decisions be made by professional cataloging personnel. A further benefit of automatic translation could be to reduce the amount of trained personnel effort required for original input. This would be accomplished by reversing the production sequence. The new process might be termed "computer-assisted editing" (see Fig. 21). The keying of the catalog data is first performed, then the computer programs incorporating the algorithms assign the basic identifying codes to the records, and finally, the trained personnel would review and correct or extend the computer actions.

The approach of the algorithms will be to identify the fields and sub-fields of the bibliographic record on a "best guess" basis. By this approach, it seems reasonable to attempt to reduce the total manual effort to one-half that required by present procedures.

FIGURE 21:
CONVENTIONAL CONVERSION COMPARED TO AUTOMATIC FORMAT TRANSLATION AND COMPUTER-ASSISTED EDITING


Convent. START
C.A.E. START
A.F.T. START
Source Catalog Card
Source Catalog Card
Existing Machine Records
Manual Pre-edit
Keyboard
Computer Translation to Input Format
Keyboard
Computer Edit to Processing Format
Computer Edit
Manual Post-edit/ correction
Proofread
Update
Stop
Update
Stop

2. Santa Cruz File Translation Program. The general function of the program (TRANSCOF) is to convert the codes for data in one format to codes in another format. The change is one of form, not content. Specifically, this program translates the Santa Cruz Library System input format into the ILR input format.

Some of the problems of format conversion are:

(1) Determination of the rules which categorize the data into a given format, at the field level.

(2) Identification of the "structural properties" of the original data (such as that of LC cards). Structural properties are the physical characteristics according to which the data as strings of symbols may be said (with some degree of probability) to group itself according to logical content and meaning. Very little quantitative information on the nature of the field content of catalog records is available. To obtain indicative data, a rough sample of 1000 LC cards was analyzed: 997 had the characteristic that the author surname (which is a logical content meaning) was uniquely separated from the remainder of the name by a comma (a physical characteristic). Thus, one trivial but useful structural property of LC card data is that the surname is separated from the given name of an author by a comma with a probability of about 99%, yielding a reliable way of confirming sub-type of personal name.

Structural properties with a high favorable probability are being used in the development of algorithms which establish equivalence between the data elements of one format with those of another. For example, using the structural property illustrated above, we developed the following algorithm:

> Given the code in the source record which identifies the field containing main entry author name, we can infer that the symbols preceding the first comma in a left-to-right character scan constitute those characters which belong to the element 'surname', and those subsequent to the comma belong to the category 'given names'. See Fig. 22.

Where structural properties are too ill-structured to describe or have an unfavorable probability, indirect methods must be developed for the division of a field in one format into the several corresponding elements of another, more specific format. Fig. 23 depicts the initial version of a sub-routine to identify the elements in the title field, a more difficult translation.

To illustrate further, consider the problem of subdividing the element 'subject headings' into sub-categories

FIGURE 22:
FLOW CHART OF PERSONAL AUTHOR FIELD ALGORITHM

Given: All of the information concerning personal author as found on an LC card, except information such as 'editor', translator', etc., which is given by a code letter in col. 69 of the Santa Cruz input card. To be determined: (1) Which portion of the string constitutes the name of the author, (2) which portion is the date date(s), (3) whether or not the title 'Sir' (which is the most common title) is present.

The following is a generalized outline of the logic involved and does not faithfully represent the actual coding of the subroutine PERSAUTH.

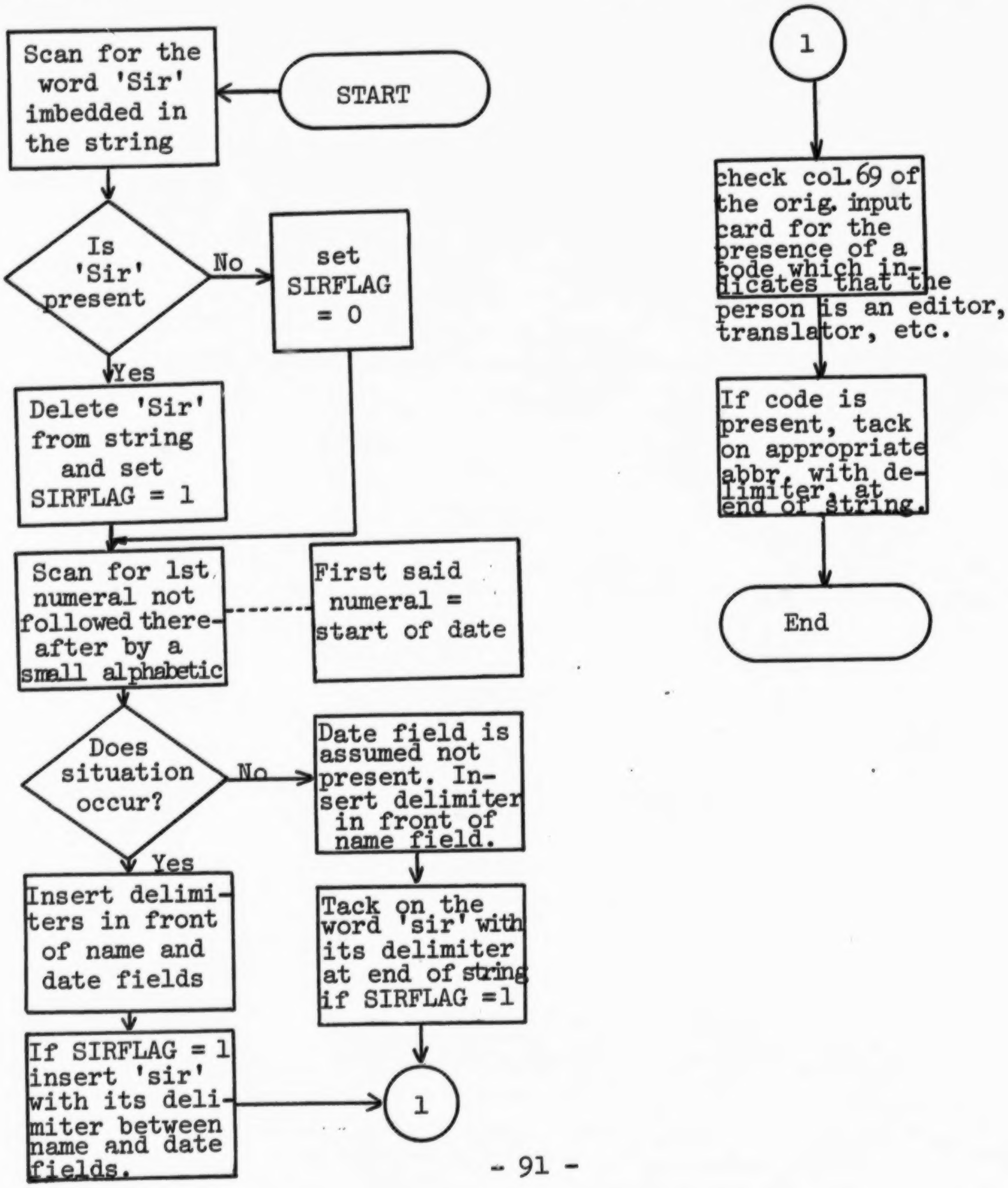

START
Scan for the word 'Sir' imbedded in the string
Is 'Sir' present
No
set SIRFLAG = 0
Yes
Delete 'Sir' from string and set SIRFLAG = 1
Scan for 1st numeral not followed thereafter by a small alphabetic
First said numeral = start of date
Does situation occur?
No
Date field is assumed not present. Insert delimiter in front of name field.
Tack on the word 'sir' with its delimiter at end of string if SIRFLAG =1
Yes
Insert delimiters in front of name and date fields
If SIRFLAG = 1 insert 'sir' with its delimiter between name and date fields.
1
1
check col. 69 of the orig. input card for the presence of a code which indicates that the person is an editor, translator, etc.
If code is present, tack on appropriate abbr. with delimiter at end of string.
End

FIGURE 23: FLOW CHART OF TITLE FIELD ALGORITHM

Given: Complete monograph title field.
To be determined: (1) Short Title, (2) long title, (3) author statement and additional information, (4) edition statement, (5) remainder of edition statement.

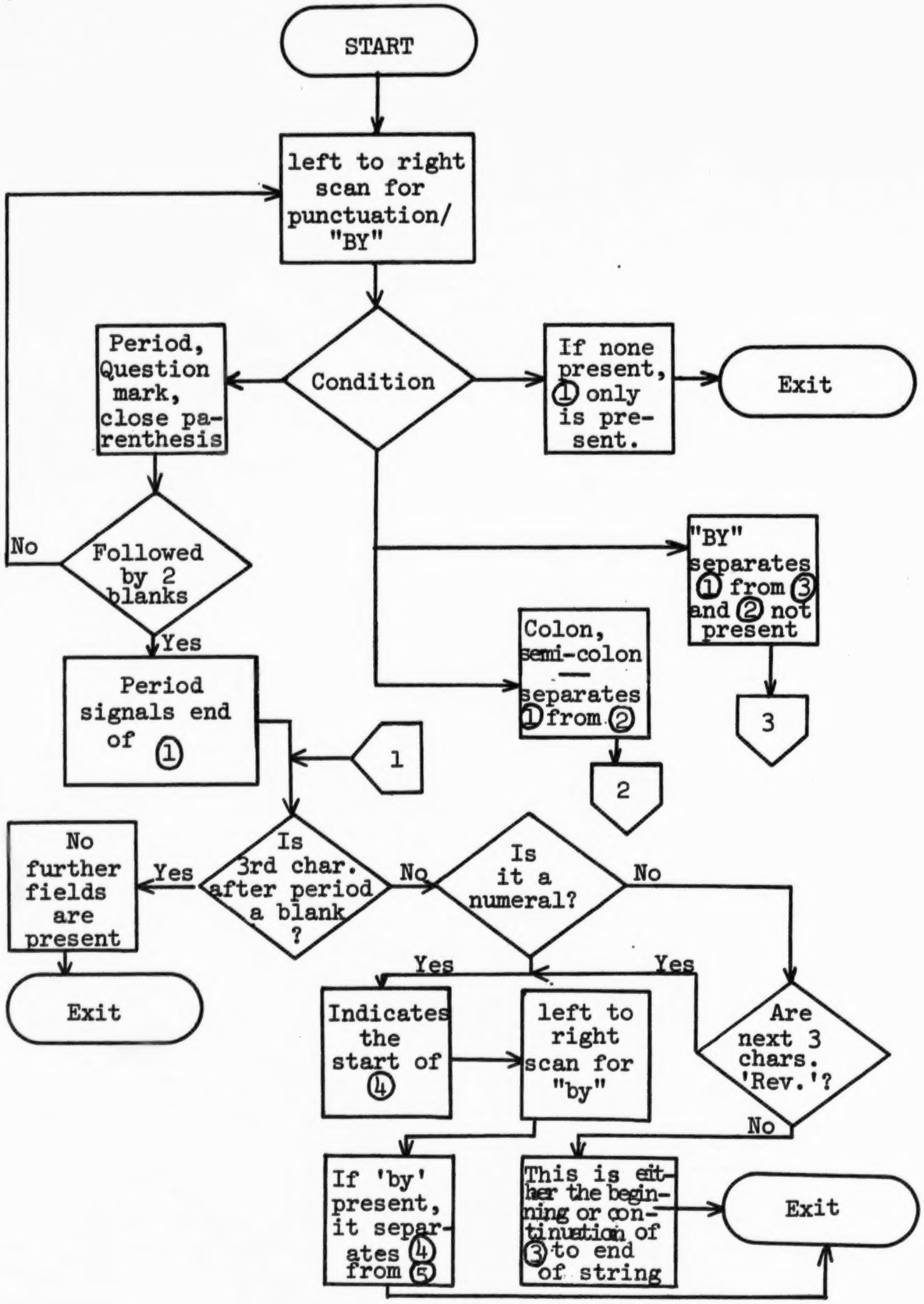
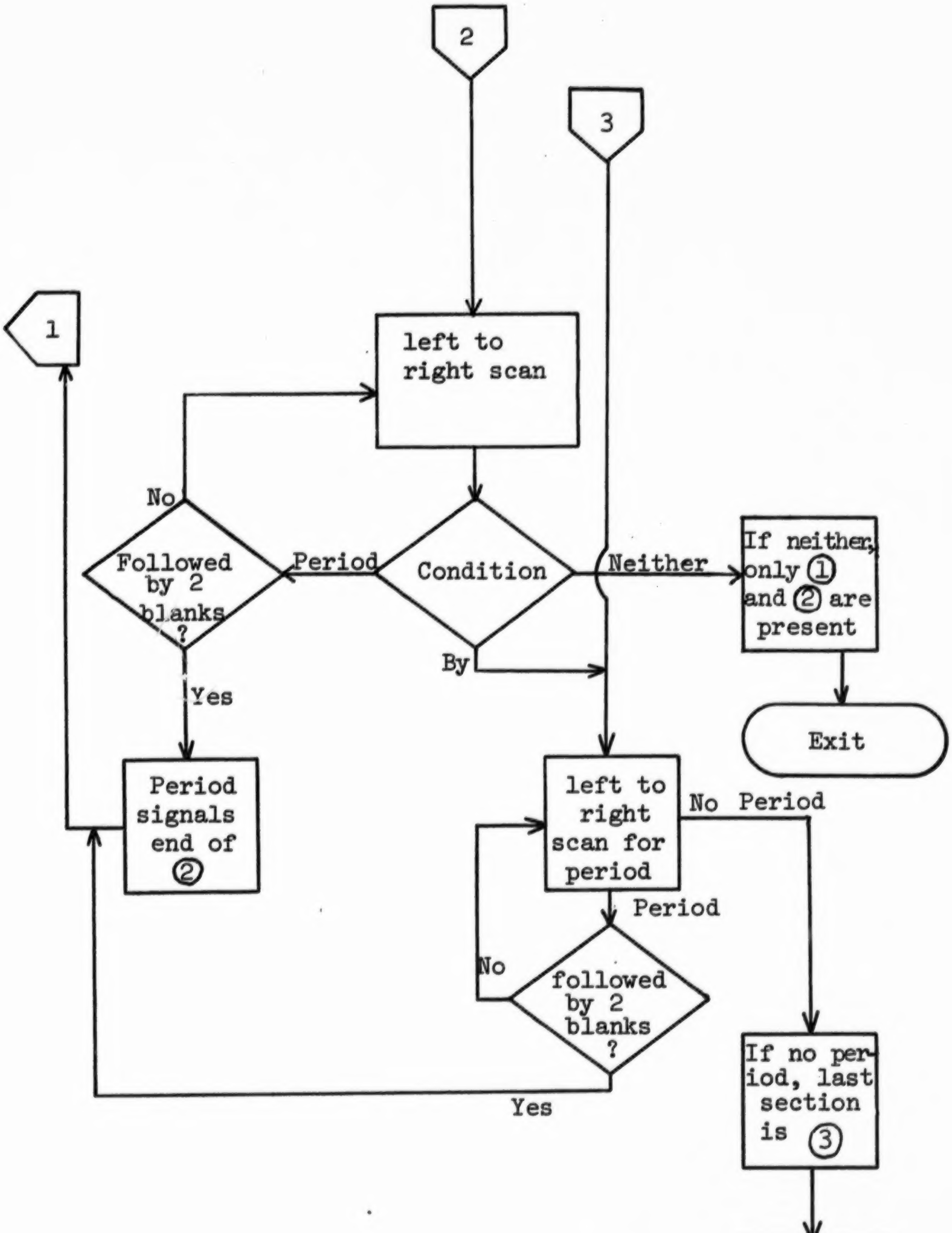

START
left to right scan for punctuation/ "BY"
Condition
Period, Question mark, close parenthesis
If none present, ① only is present.
Exit
No
Followed by 2 blanks
Yes
"BY" separates ① from ③ and ② not present
Colon, semi-colon — separates ① from ②
Period signals end of ①
1
2
3
No further fields are present
Yes
Is 3rd char. after period a blank ?
No
Is it a numeral?
No
Yes
Yes
Exit
Indicates the start of ④
left to right scan for "by"
Are next 3 chars. 'Rev.'?
No
If 'by' present, it separates ④ from ⑤
This is either the beginning or continuation of ③ to end of string
Exit

FIGURE 23 (Cont.):
FLOW CHART OF TITLE FIELD ALGORITHM


2
3
1
left to right scan
Condition
Period
Neither
By
No
Followed by 2 blanks ?
Yes
Period signals end of ②
If neither, only ① and ② are present
Exit
left to right scan for period
No Period
Period
No
followed by 2 blanks ?
Yes
If no period, last section is ③
Exit

based on the various classes of subjects, corresponding to the 6XX tag series in MARC II. There is nothing in the structure of information on the LC card which provides any clue to its meaning or type, beyond the level of "subject tracings" identified by Arabic numerals. Resort might be made in this case to table-lookup scheme utilizing a dictionary, which identifies the field type to which a given subject heading belongs. A "tagged" version of the LC subject headings machine file could be the basis for such a method.

The TRANSCOF program consists of a main routine called LAYOUT and five subroutines named PERSAUTH, CØRPAUTH, ES200500, TITLEALG and PPD.

The main program performs three functions, in three sections:

(1) Reads in the set of 'card images' from a source tape that make up the data converted from a single catalog card. This data is then broken down and laid out into the appropriate categories for processing by the subroutines. Certain characteristics of the data are noted and recorded for later use. Branches are made to other sections of the main program upon detection of variations in the data (such as missing sections) for "troubleshooting" purposes.

(2) The subroutines are called and various logical gates are opened or closed in the third section.

The subroutines apply the algorithms to the appropriate strings of data which were constructed in the first part of the main program. Data which is input to the subroutines are restructured into the ILR input format.

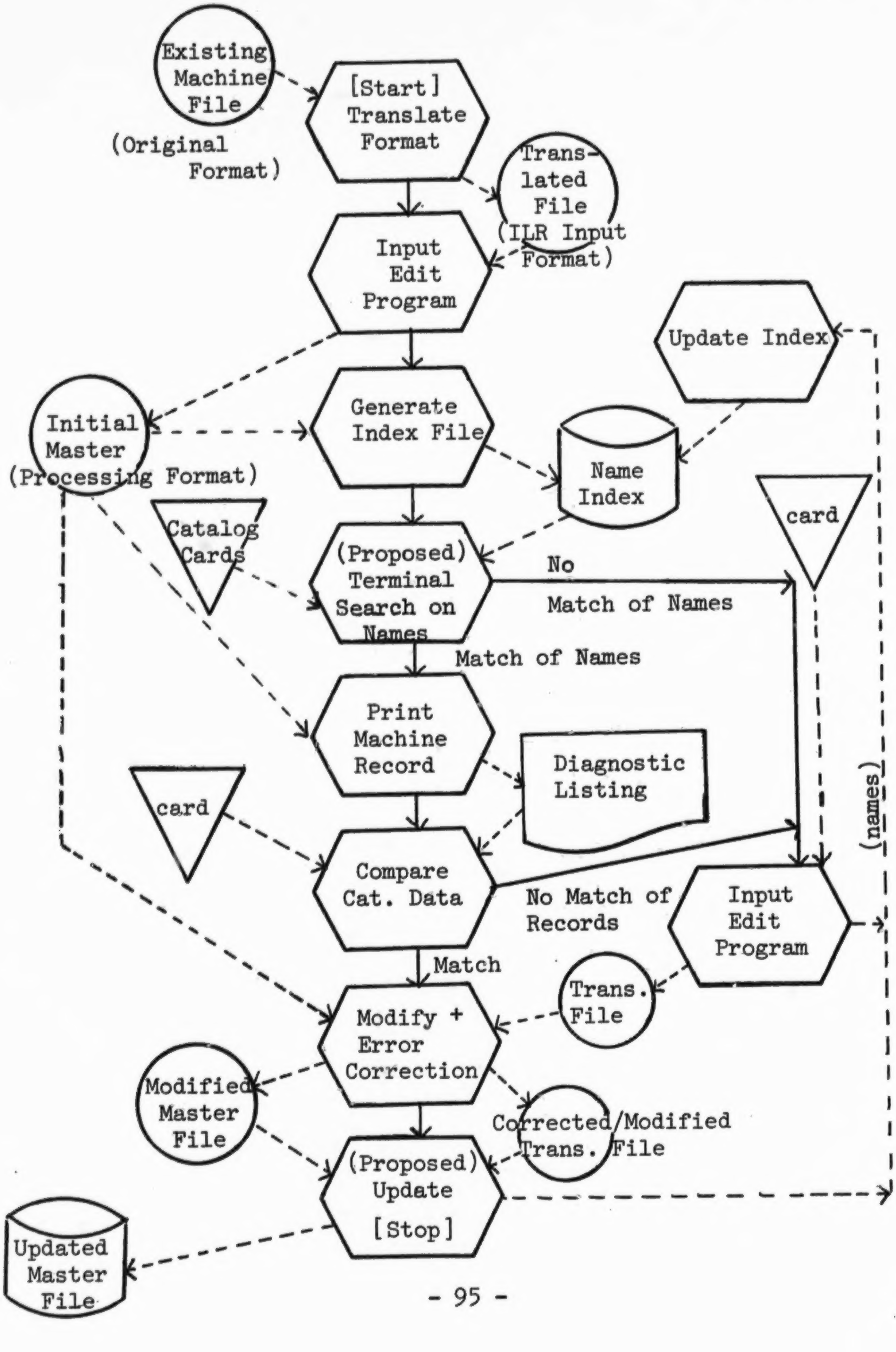
(3) The various pieces of the restructured data are concatenated to produce a continuous string which makes up an ILR input record. This is then to be written to tape, and the whole process is repeated for another record.

D. DATA BASE PRODUCTION PROCEDURE

In the following section we present a detailed plan of the procedure through which we plan to obtain the data base for the study.

The chart in Fig. 24 summarizes the overall file creation plan, including the experimental on-line, terminal-based search operation. Certain aspects of the process have not yet been implemented as of this writing. These components are labelled "Proposed". Two programs in this plan are critical and were discussed in preceding sections. These are

FIGURE 24: SUMMARY CHART OF DATA BASE PRODUCTION


Existing Machine File
(Original Format)
[Start] Translate Format
Translated File (ILR Input Format)
Input Edit Program
Update Index
Initial Master (Processing Format)
Generate Index File
Name Index
Catalog Cards
card
(Proposed) Terminal Search on Names
No Match of Names
Match of Names
Print Machine Record
Diagnostic Listing
card
Compare Cat. Data
No Match of Records
Input Edit Program
(names)
Match
Trans. File
Modify + Error Correction
Modified Master File
Corrected/Modified Trans. File
(Proposed) Update [Stop]
Updated Master File

the program for conversion of the Santa Cruz tapes to ILR input format (TRANSCOF) and the input edit (processing format) program (INFOCAL). A preliminary version of the INFOCAL program is currently running.

Meanwhile, the modules of the system were set up in a "stand-alone" status. To implement the program for original input separately, a large sample extracted from the source file of University of California Catalog Supplement was used. The material in this sample was edited, keypunched, and processed to shake down the record formats and to serve as a basis for evaluating alternative methods of converting existing machine files. That is, we plan to compare the records put through the Santa Cruz format translation with any matching records which are input as original records. This will serve as a basis for improvement and revision of the automatic format translation program.

1. Proposed On-line Search Procedure. Instead of using manual methods to verify which catalog entries have been converted to machine form, we plan to use a terminal-based search system for this purpose. The search key to be used in this experiment would be the main entry author name (or a portion thereof) from the card, matched against the machine index file of author names. A research objective of this technique, as well as to shorten the search process, is to gather quantitative data on the optimal length of search key, i.e., minimum number of characters in the name to be keyboarded in the request while preserving uniqueness (see Section II.E.4).

If there is a match, the catalog card would be set aside to be checked against a subsequent printer listing of the full machine record. If no match is obtained, the catalog card would be put immediately into the conversion process. At the same time, the index file would be updated with the new information indicating that the entry is in the conversion process but not yet in the system. As a separate manual process the printed listing would later be checked against the catalog entries which were set aside. If a true match is confirmed, the added location data or other modifying information such as variant call number would be incorporated into the existing record. If the possible match is not true, the catalog card would then be put into the normal conversion process as a new record.

Fig. 25 shows the first steps in this experimental verification search process.

Fig. 26 shows the cards found to be possible matches with existing machine index file entries being compared with printouts for corresponding machine records from the computer master

FIGURE 25: ON-LINE SEARCH FOR DUPLICATES

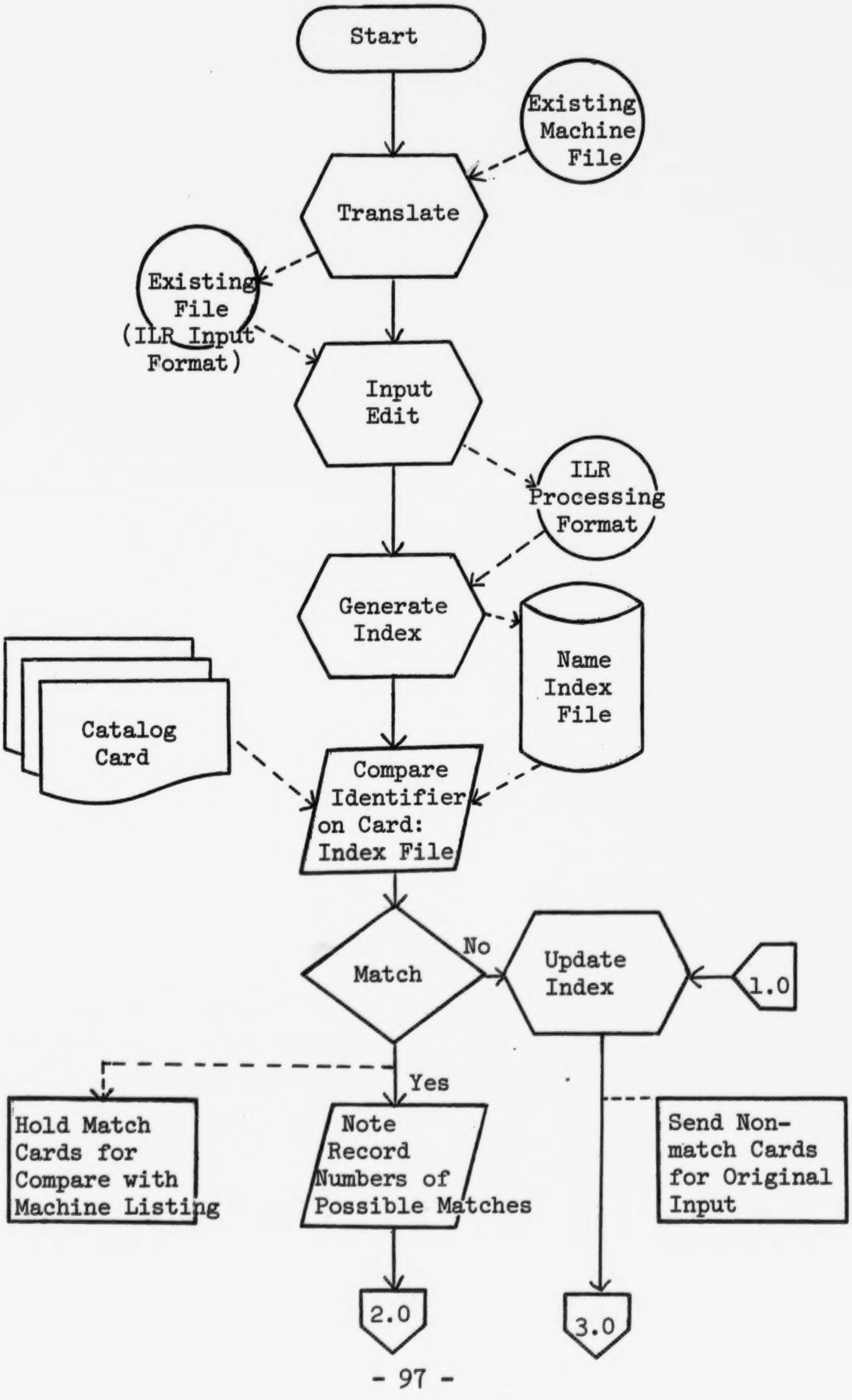
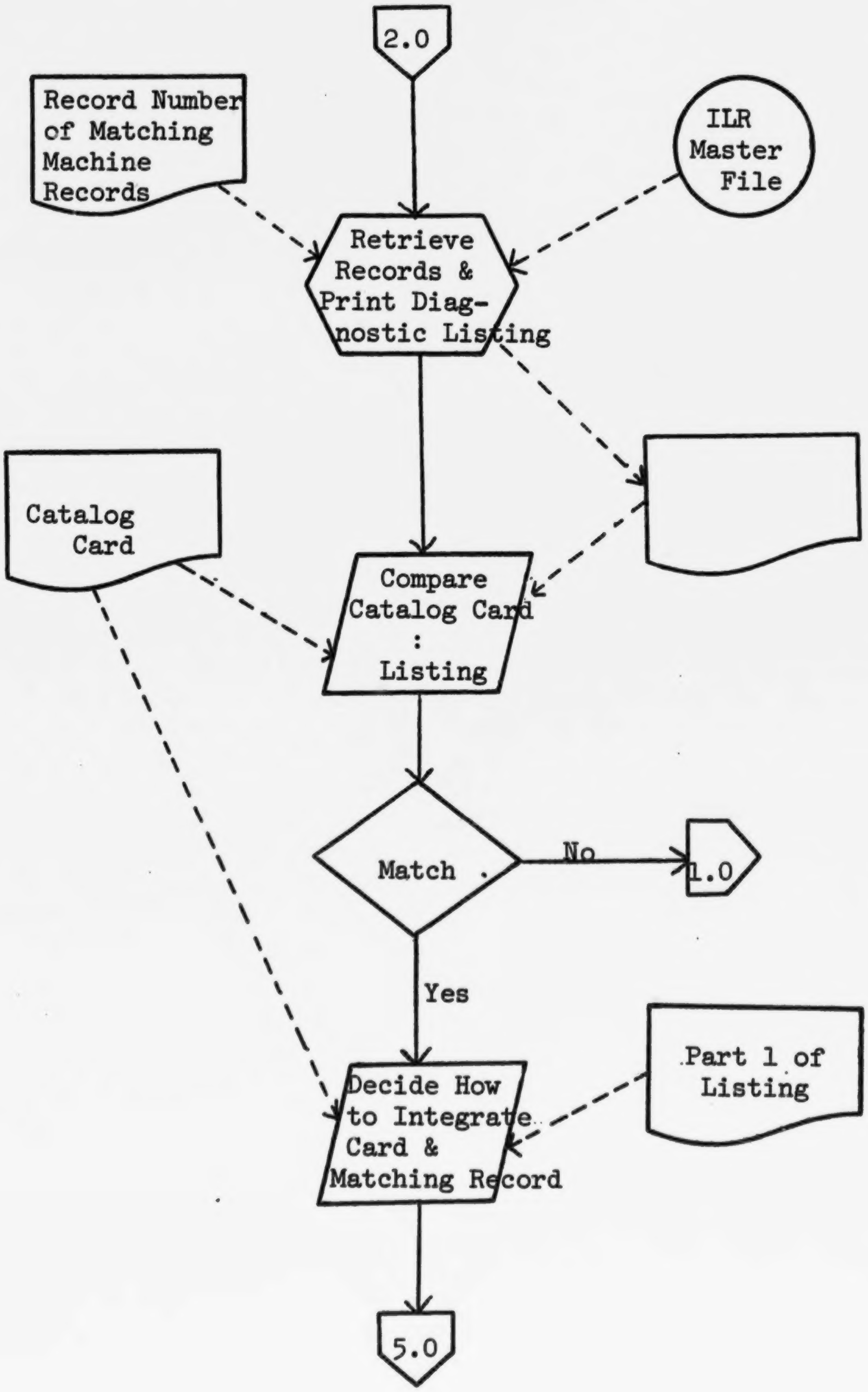

Start
Existing Machine File
Translate
Existing File (ILR Input Format)
Input Edit
ILR Processing Format
Generate Index
Name Index File
Catalog Card
Compare Identifier on Card: Index File
Match
No
Update Index
1.0
Yes
Hold Match Cards for Compare with Machine Listing
Note Record Numbers of Possible Matches
Send Non-match Cards for Original Input
2.0
3.0

FIGURE 26: VERIFICATION OF MATCH


2.0
Record Number of Matching Machine Records
ILR Master File
Retrieve Records & Print Diagnostic Listing
Catalog Card
Compare Catalog Card : Listing
Match
No
1.0
Yes
Decide How to Integrate Card & Matching Record
Part 1 of Listing
5.0

file. If a verified match is confirmed, the editor decides how to merge the manual record with the machine-readable information. If no true match is found, the name heading on the catalog card is entered into the machine index file as a control on the new record, which now goes into the original input process.

2. Editorial Preparation. The next chart (Fig. 27) shows the process of editing and initial keyboarding. The cards flowing into this stage are those which did not match, either on the initial name (author) search, or which were found on actual comparison with a printout from the machine file to be distinctive records, even though the heading matched.

If the records matched on name and also matched on title, edition, etc. when the card was compared with the printout, the variant data from the catalog card is input as a correction to the machine record, as depicted in Fig. 28.

Several thousand records have now been edited in the ILR Input Format. This work is under evaluation in order to establish the adequacy both of the formats and of the organization for providing the input.

Preliminary estimates of the effort requirements will be used as a basis for evaluating the results of the first year's editing and for reorganizing the input methods for the remaining portion of the current data base preparation work. Prior experiments indicate that the input (including all steps of editing, keyboarding and correcting) of English language material should be capable of being performed with an approximate total of 5 man-minutes of effort per title.* Our recent work using the ILR input format that translates to full MARC II format indicates that about 15-20% more effort per title will be required, at least until the editing and typing staff reach a high degree of proficiency. It is estimated that non-English material will require on the order of between one-half again to two times as much effort as the English language material.

3. Keyboarding. The records are next sent for keyboarding. During initial tests in ILR the production rate achieved was somewhat lower than that attained in previous studies. This is due to the impact on our coding of the

---

*Cartwright, Kelley L. and R.M. Shoffner. Catalogs in Book Form: A Research Study of their Implications for the California State Library and the California Union Catalog, with a Design for their Implementation. Berkeley, Institute of Library Research, University of California, 1967. p. 48-49, I-31.

FIGURE 27: DATA PREPARATION AND TRANSCRIPTION

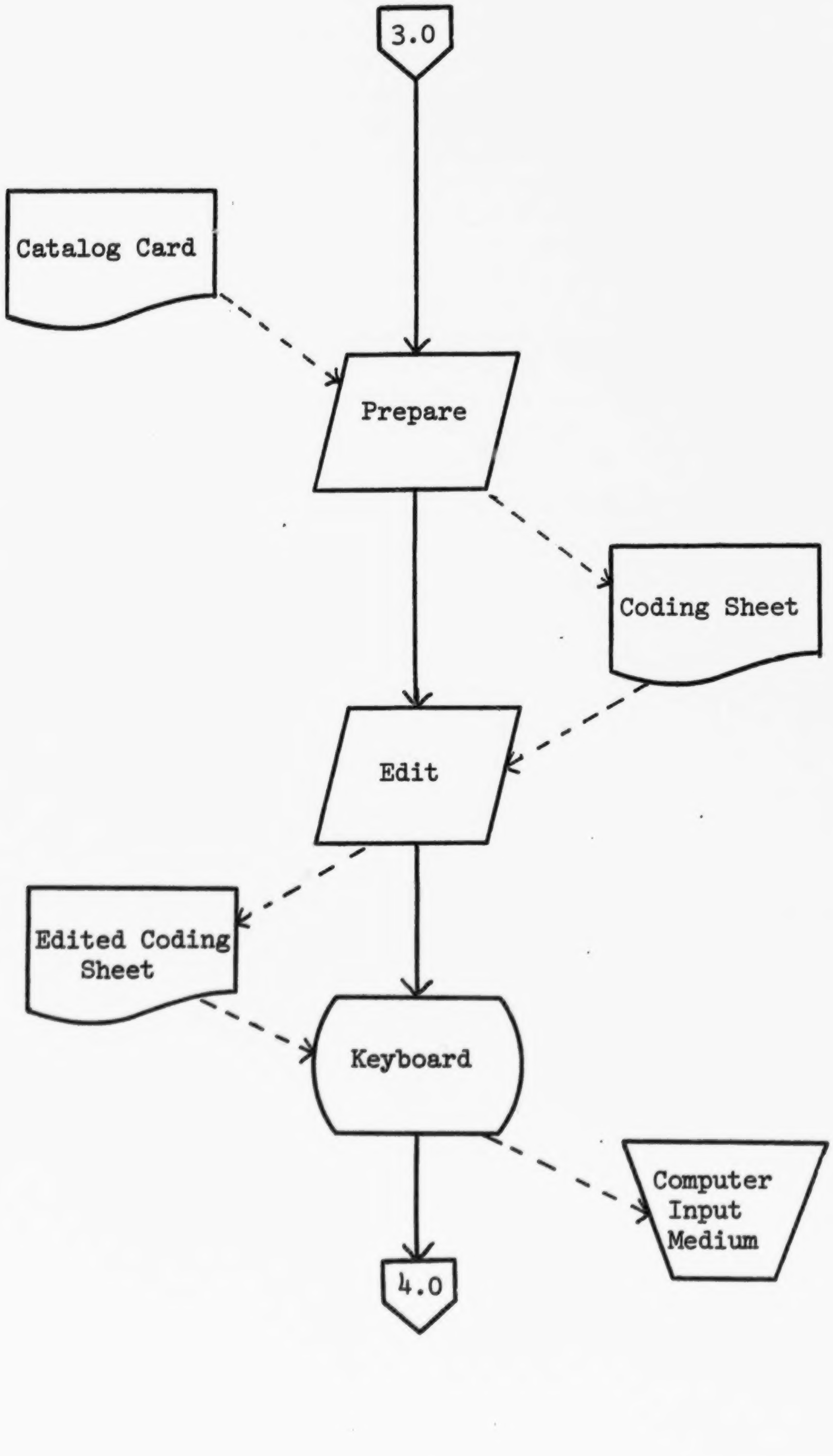
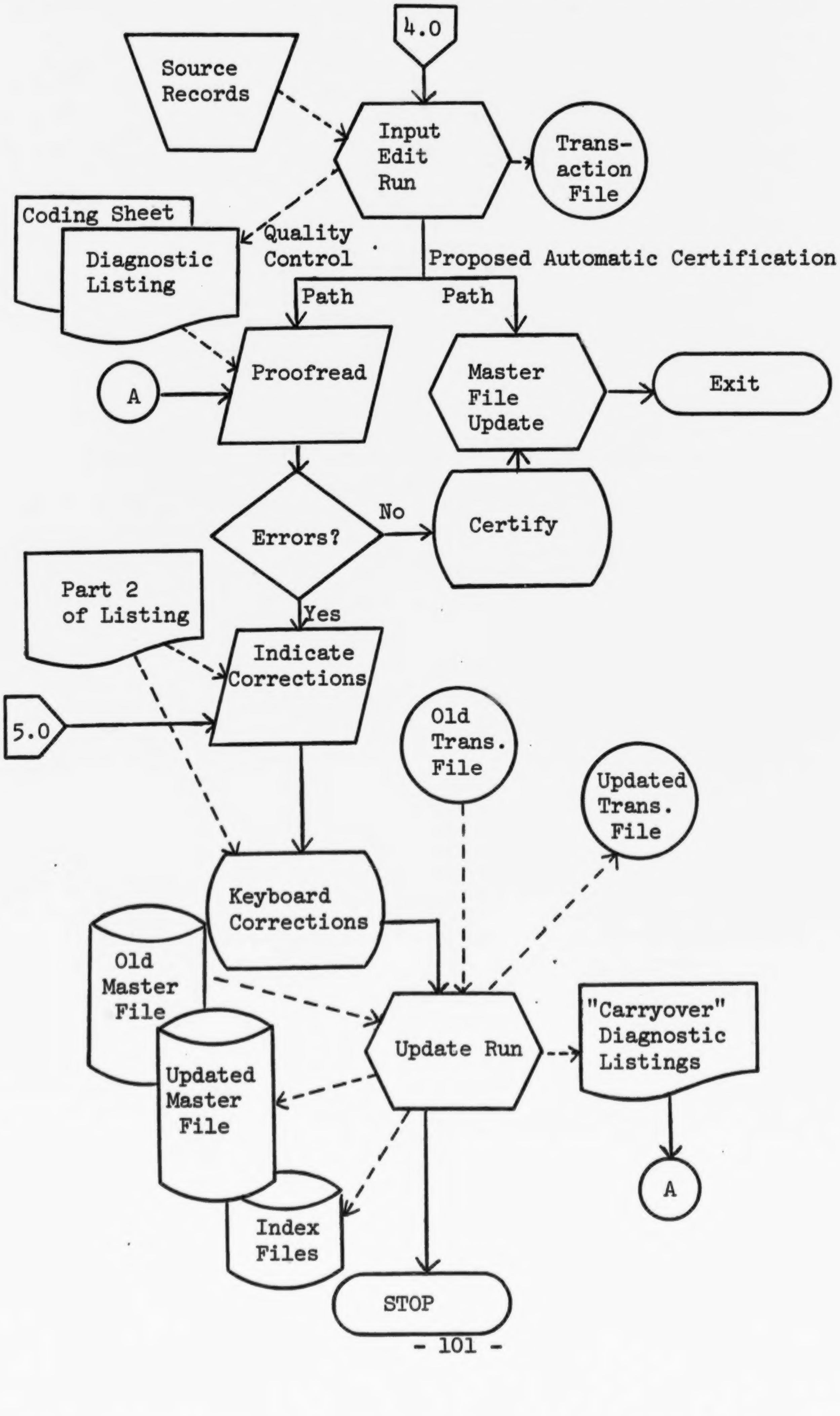

3.0
Catalog Card
Prepare
Coding Sheet
Edit
Edited Coding Sheet
Keyboard
Computer Input Medium
4.0

FIGURE 28: COMPUTER EDIT, CORRECTION CYCLE, AND FILE UPDATE


4.0
Source Records
Input Edit Run
Trans-action File
Coding Sheet
Diagnostic Listing
Quality Control Path
Proposed Automatic Certification Path
A
Proofread
Master File Update
Exit
Errors?
No
Certify
Yes
Part 2 of Listing
Indicate Corrections
5.0
Old Trans. File
Updated Trans. File
Keyboard Corrections
Old Master File
Updated Master File
Update Run
"Carryover" Diagnostic Listings
A
Index Files
STOP

added complexity of the MARC II format, primarily. Preliminary estimates secured from commercial service bureaus for the initial keyboarding on a production basis are for a rate of about 5,000 characters per hour for the learning phase, increasing to a rate of 7,000 characters per hour after that, on the average. This is based on the input record as described in this report.

A number of keyboarding conventions are being recommended for use with the lengthy, complex strings involved in catalog records. Among these are a "null previous character" signal, i.e., a code that can be typed immediately following a character typed by mistake (when so recognized by the device operator), and a "spacing rule". The device operator will not have to be especially attentive to inter-word spacing when inputting the catalog data since an internal computer edit routine will check the data stream for instances of more than one space and reduce such occurrences to one space.*

---

*There is one exception which is very important if the input process happens to be implemented on tab cards. That is to employ a convention that a punched card can be either filled completely (i.e., a character in the last column) or partially filled. If partially filled, all trailing blanks (from the last character punched to the last column available for use) will be automatically purged from the input stream by the edit program. In any case, the first normally used column of the next succeeding card must contain a blank if there is an inter-word space needed. If not needed, the following card begins with the next succeeding character of data in the first normally used column, as in the case of a break in a word at the right end of the card being punched.

Strict conformity to this or a similar arbitrary convention is absolutely required to make the input record usable. Irregularity in spacing can create great difficulty in translation of a machine file from one format to another, particularly in cases where tab cards were used as the conversion medium.

4. Proofing and Correction. The records are then input to the computer, and the internal processing record is compiled by the input edit program. (See Fig. 28). After a number of machine edit checks, validations, etc., are performed, a diagnostic proof-listing is printed out. This printout here is optional: it may be suppressed if the record is not to be selected for proofing by the statistical quality control routine. A line printer will be used rather than a slower, more costly on-line terminal.

The kind of validation checks performed are: check to determine whether ten slashes have been inserted into the record; signal an error condition if less than ten are present. Check detailed aspects of the coding, such as whether author-analytic added entries contain both an author and a title subfield, as required by the input instructions. On the double-spaced listing to be used by the proofreader, the computer prints out error messages from a repertory of such diagnostic routines. These messages appear in conjunction with the data containing the detected error. Examples of printed error messages generated by the edit programs are:

DECKLET SEQUENCING ERROR. The card numbers in the set of input record tab cards (if that is being used for input) are not in ascending numeric order.

cc IS INVALID ADDED ENTRY TYPE. "cc" is a two-character code describing the type and sub-type of a heading. The code designated was not recognized by the program as a valid code, indicating an editor error or typographical error was committed.

MORE THAN 3 BIBLIO. LEVELS. EXTRA CODES IGNORED. Only three positions are allocated in the record leader for storing the values for bibliographic levels of the record.

TRANSLATION INDICATOR GIVEN, BUT 2ND LANG. MISSING. The language variable field contained only one three-character language code: there should be two.

ONE OR MORE A-FIELDS MISSING. ____ SLASHES FOUND. This message is printed if the program finds more than or less than 10 slash marks between the location of the beginning of the A-fields and the location of the start of the B-fields.

IMPROPERLY CODED LOCATION SUB-FIELD IN HOLDINGS STATEMENT. ____***. The sub-field did not contain a five-digit agency code or a five-digit-plus-one-letter sequence.

WARNING - LC CALL NUMBER OCCURRED MORE THAN ONCE. This signals the proofreader that the field code for LC call number was used more than once. Could occur when proper code for LC copy statement was not used for the second call number.

There are a number of error conditions which cannot be determined by the computer, and these would of necessity have to be detected in the visual proofing process. In case an error goes into the master file undetected, a procedure will be established to input corrections reported by users at a later time.

Two features of the correction cycle are of particular interest: 1) the re-keyboarding of the corrections could be performed experimentally via the on-line terminal, in contrast to the initial keyboarding which is done off-line on devices such as the keypunch, and 2) the diagnostic listing of the record is being experimentally printed out in two parts, one containing the coding and data displayed in a logical field array, element by element, field by field; the second part listing the same data but in order corresponding to the sequence of the physical input record, e.g., a tab card decklet.

This form is an experiment intended to assist the proofreader. The first part of the listing is shown in Fig. 29, and is used by the proofreader (who will not be the same person as the editor of any given record) for visual scanning, comparison with the coding sheet, and detecting of errors.

The second part of the listing (See Fig. 30) will be used for re-keyboarding: the proofreader will transcribe keying instructions on the card image listing itself, and the marked-up listing will be transmitted to the correction typist, who will input the corrections into the file. The operator scans the listing for the indicated errors to be corrected, and re-keys only the amount of information necessary to correct those errors. This information will consist, first, of the information which is in error, and, second, of the correct information as it should be. Each line of the print-out will have a number, which is assigned by the computer. When the computer is given the incorrect information which occurs in that line, it will search the corresponding internal line to find that information, and then change it to the indicated correct information. This technique avoids the necessity to re-key either entire print lines or entire physical units such as punched cards.

Such a "string correction routine" can be set up to process errors at the level of single character, words, lines, groups of lines, or whole records. It can be operated through any input device, either on-line or off-line.

```
                                                RECORD NUMBER   100150

1966dmdsebfa690ja/%%690r%jbBF435.G193c%%430%manaw00

/R1.A52 v.40no.4supp.

/Garrison, Mortimer,%%%ed.

/Cognitive models and development in mental retardation,

/%edited by Mortimer Garrison, Jr.

/New York,

/American Association on Mental Deficiency,

/1966.

/149 p.

/

/(Monograph supplement to American journal of mental deficiency, v.70, no.4, 1966)

*k"Proceedings of a Research Conference sponsored by the Woods Schools, the American
Association on Mental Deficiency and the National Institute of Child Health and Human
Development"

+
```

FIGURE 29: DIAGNOSTIC PRINTOUT, PART 1 - LOGICAL FIELD LISTING

```
                                        FILE ORGANIZATION PROJECT

                                           DIAGNOSTIC LISTING
DATE    05/02/68
                                                                  RECORD NUMBER    100150

  159       100150  01       1966DMDSEBFA690JA/%%690R%JB_B_F435._G193C%%430%MANAW00/_R1._A52 V.40NO.4

  160       100150  02       SUPP./_GARRISON, _MORTIMER,%%%ED./_COGNITIVE MODELS AND DEVELOPMENT IN M

  161       100150  03       ENTAL RETARDATION,/%EDITED BY _MORTIMER _GARRISON, _JR./_NEW _YORK,/_AME

  162       100150  04       RICAN _ASSOCIATION ON _MENTAL _DEFICIENCY,/1966./149 P.//(_MONOGRAPH SUP

  163       100150  05       PLEMENT TO _AMERICAN JOURNAL OF MENTAL DEFICIENCY, V.70, NO.4, 1966)*K"_

  164       100150  06       PROCEEDINGS OF A _RESEARCH _CONFERENCE SPONSORED BY THE _WOODS _SCHOOLS,

  165       100150  07        THE _AMERICAN _ASSOCIATION ON _MENTAL _DEFICIENCY AND THE _NATIONAL _IN

  166       100150  08       STITUTE OF _CHILD _HEALTH AND _HUMAN _DEVELOPMENT"+
```

FIGURE 30: DIAGNOSTIC PRINTOUT, PART 2 -
CARD IMAGE LISTING

Following this correction of the initial product, it is necessary to proofread the corrections made by the computer, in order to determine that these have been made correctly, and that error has not been introduced in the second keying. In the second proofreading, only the corrections indicated by the proofreader and the corresponding actions taken by the computer, will be compared to one another. It will usually not be necessary on the second proofreading to compare again the information printed by the computer, to the full information on the original catalog entry.

The final step is to update the machine files with the new record.

E. ISSUES OF COST AND QUALITY

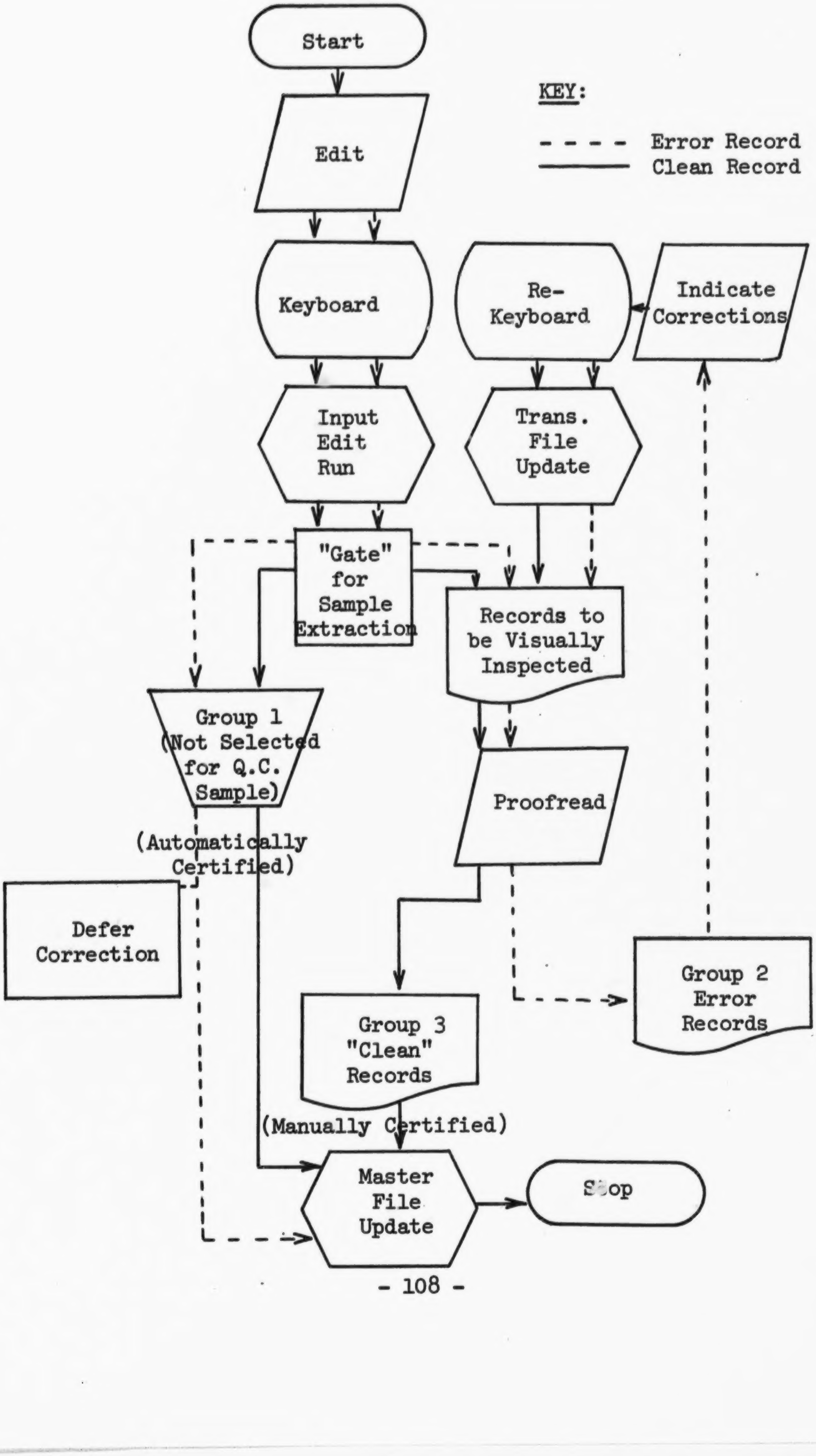
1. Data Conversion Quality Control. The data preparation system is composed of four relatively independent processes: editing, keyboarding, visual proofreading, and error correction (re-keyboarding).* Fig. 31 shows the flow of both "clean" records and defective records through the quality control subsystem. Distinction will be made among three sources of an error: editing, keyboarding, and computer processing (program errors, etc.). The detailed attribution of error is for managerial purposes (e.g., revision of procedures and editor training), and to prepare billing of a keyboarding contractor, if the conversion were to be done by an outside service.

Error records (dashed lines) originate mainly in the editing step and the initial keyboarding step. To detect and tally errors introduced by either of these sources, a control element is introduced into the flow.

---

*This procedure does not include a control step called "proofing after editing", wherein the editors exchange coding sheets among each other prior to initial keyboarding. This was thought to be an unwarranted extra cost in view of the high calibre of the staff available for editing. Also excluded from the above is the review of trainee editors' production.

FIGURE 31: QUALITY CONTROL SUBSYSTEM


Start
KEY:
Error Record
Clean Record
Edit
Keyboard
Re-Keyboard
Indicate Corrections
Input Edit Run
Trans. File Update
"Gate" for Sample Extraction
Records to be Visually Inspected
Group 1 (Not Selected for Q.C. Sample)
(Automatically Certified)
Proofread
Defer Correction
Group 2 Error Records
Group 3 "Clean" Records
(Manually Certified)
Master File Update
Stop

Instead of making this a separate stage, it seemed most feasible to couple the control process with the proofreading step, by modifying the latter. "Modified proofreading" would include the normal processes of error detection and marking for the purposes of re-keyboarding. It would also include tallying of the errors detected and their attribution to editing, keying, or computer program. An advantage to this procedure is a large reduction in the cost of the sampling needed to establish and maintain the quality control procedure itself.

The basic decision is that not all records need be exhaustively proofread immediately after initial computer input, as has normally been the case in library conversion. Instead, the editing and keyboarding of all records, and the modified proofing of only some of the records (or, as a separate option, proofing of certain critical parts of each record) is performed as one sequence. The exhaustive proofreading of the remainder of the records (or the remaining parts of all records) and the correction of all of the errors can be deferred and performed at a later time. (There will be a procedure for correction of errors reported by file users, of course.)

The number of records to be proofread in the first cycle of the control procedure will be that sample size required to obtain the desired confidence level of the information to be fed back in regard to keyboarding and other errors. If the error rate is high at the beginning of the production operation, as is likely, then the number of records to be proofread will be high since one of the variables governing the size of the samples selected for inspection is the number of defective records found in the initial sample. (A procedure for sample size determination is included in Appendix VI.)

The output of the initial cycle would be three groups of records. Group 1 is those records not selected for proofreading on the initial cycle. These would be automatically transferred to the master file, and can be inspected at a later date in a systematic manner for purposes of correction. Group 2 is that set of records, selected for proofing, which have been found to contain errors and are now ready for re-keyboarding. Group 3 is that set of records, also selected for proofing, which have been found to be error-free and are ready for transfer to the master file. This is an iterative process, i.e., Group 2 records are continuously re-cycled until they are removed from the process by falling into Group 3.

2. A Decision Model for Keying Cost/Quality. In planning for a large scale conversion it may be desirable to contract out certain of the tasks which are one-time, non-continuing in nature. The keying of the data is a portion of the effort which might be done by an outside group. However, a method for deciding how to select among alternative conversion systems and prices for data transcription service is needed. The requirement is to secure the greatest amount of converted data at the least expenditure both of time and money. Also, the quality of the conversion is vital. We suggest that a solution is not found in simply choosing the method or firm which offers a given level of quality for a fixed price (time and materials). Rather the variables must be related to each other.

In the circumstances of this project, one of the questions asked is "What will be the cost impact on the record correction part of our production, of a given level of accuracy of the data obtained in the initial keyboarding?" This issue underlies several stages of the process, in particular the level of effort for re-keyboarding. We have generalized our decision making scheme in the hope that it might be useful to other organizations facing similar questions concerning conversion.

The effective conversion cost for a unit record of input is that needed to attain an error-free record. This accuracy can be defined by degrees of approximation to perfect data, that is, to 100% accuracy.

Given that the error rate of the initial keyboarding is low enough, one may choose to accept the records without correcting them (at this time). In choosing between alternatives with different costs and error rates one should select the one with the lowest cost for the accurate records, where this cost is computed by allocating all conversion cost to the accurate records. If we characterize the error rate as E_i, then we can compute an effective cost per record, \$AR, as

$$\$AR = \frac{\$SB}{1 - E_i/100} \qquad (1)$$

where \$SB represents the initial keying cost per record.

If a higher level of accuracy is needed than can be provided during initial keying, then subsequent proofing and correction will be required. The more accurate the initial keyboarding is, the less the re-keyboarding cost incurred in the correction module. Therefore, each increment of added accuracy of initial keying has a value to the organization in the replaced cost of re-keyboarding the corrections.

When we include the cost of proofing and correction in the total cost for converting data, the cost can be defined as:

$$K = \$IH + \$SB + \$ED + \$ECE_i \qquad (2)$$

where

K = total unit cost to input a record

$\$IH$ = cost of editing and associated pre-keyboard processes, per record

$\$SB$ = cost per record for initial keyboarding*

$\$ED$ = cost of visual error detection (proofreading)

$\$EC$ = cost of re-keyboarding, per error "character-block"

E_i = the number of error "character-blocks" per record. A character-block is defined as 5 contiguous characters which must be re-keyboarded to correct one or more characters within that block.

To state the cost of error in relation to the initial keyboarding costs, rearrange the equation:

$$\$SB = K - \$IH - \$ED - \$ECE_i \qquad (3)$$

With this equation, we can define a family of keying cost/error rate indifference curves by allowing the total cost, K, to take on different values. As an example, if $\$IH + \$ED = \$0.28$ and $\$EC = \$0.05$ then the variation in given error rates (E_i) sustained, can be used to establish a curve for the relation of the variation in keying cost such that K is constant:

$$\$SB = K - \$0.28 - \$0.05E_i \qquad (4)$$

Fig. 32 shows an indifference curve for the range of values of E_i and $SB.

---

*Key verifying is excluded since it doubles the cost of the initial transcription but does not result in detection of errors attributable to editing and computer processing - or even all the initial keying errors.

In Fig. 32, any price quotes for initial keyboarding that fall in the region to the right of the indifference curve would be unacceptable in comparison to any prices falling on the curve or to the left of it.

There may be some initial error rate, E_i, that is felt to to be too great to be acceptable even though the records could be corrected. Thus, we show this "unacceptable region" extending horizontally at an error rate of 10 blocks per record. Fig. 32 was shown with the values of $SB plotted in terms of K. The leftmost indifference curve identifies the best of a set of different price/error rate alternatives. To see this, consider Fig. 33, which shows three separate price quotes for initial keying, expressed as sliding scales based on error level. Part of the decision in this case is clear. Regardless of the specific accuracy obtained, "C" should not be chosen because its price is always higher than those of A or B. Thus, it would be irrational to pay price "C" under the conditions shown in the example.

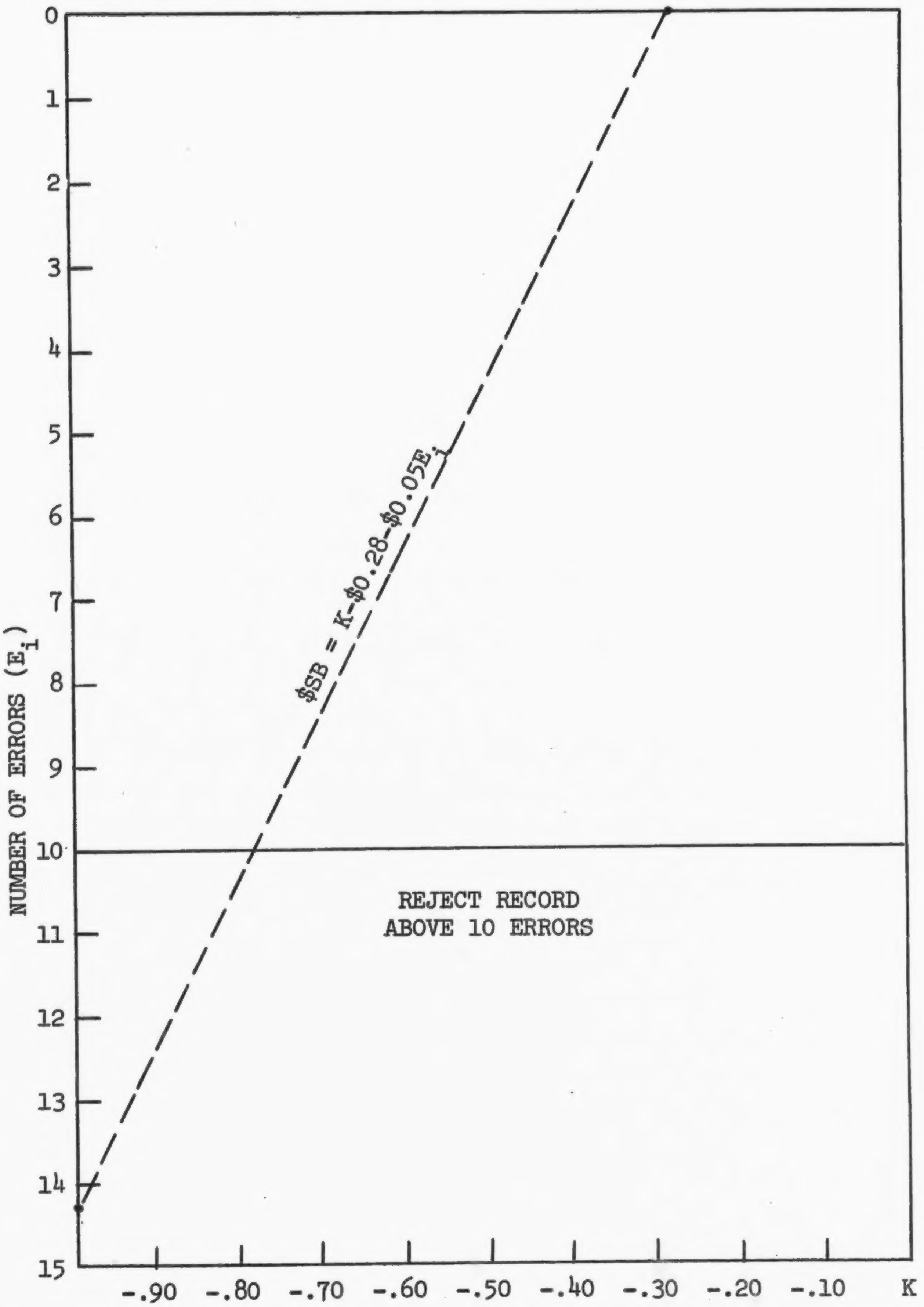
To determine which of the two prices, A or B, should be chosen, the indifference curve should be moved to the left as far as possible and yet allow it to lie upon a point within the acceptable region, quoted for A or B.

The leftmost curve will intersect Price "A" at an error rate of 2. Thus, it is the preferable alternative.

In most situations a final error rate which is greater than zero is acceptable. This does not change the basic analysis, however. Rather it defines an error rate below which (plotted above on our figures) additional cost will not be justified. The leftmost indifference curve which intersects a price/error rate curve in the given error region of interest will indicate the best alternative. In the illustration, a final error rate of 2 might be acceptable. The price curve "A" would still be the one for the organization to select.

Thus, using the level of accuracy desired in initial keying, at a given keying cost as the critical variable, we have established a basis for selecting among alternative costs for keyboarding. This error rate is to be determined by visual inspection of input records selected by the method of sequential acceptance sampling outlined in the preceding section.

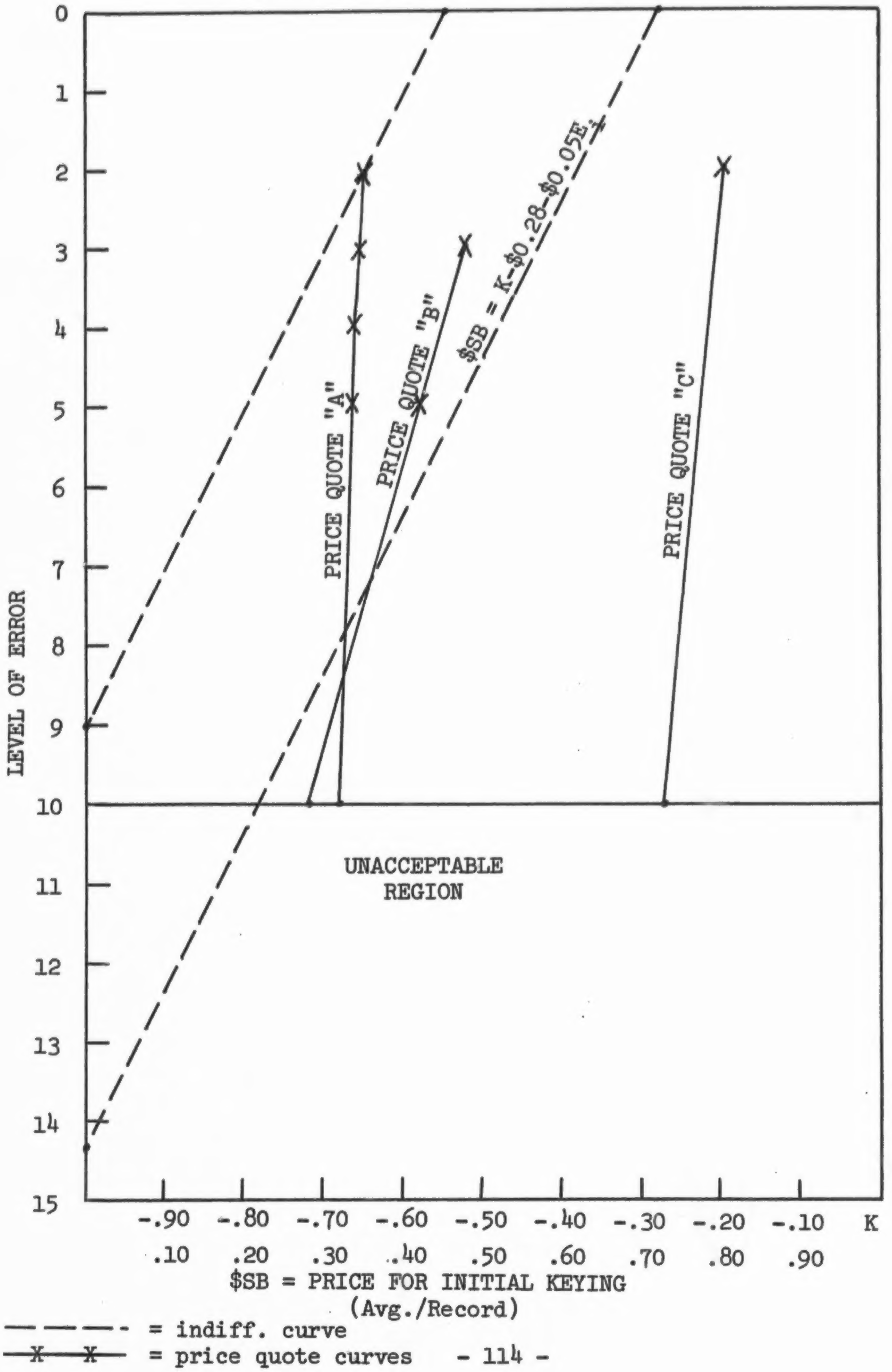
FIGURE 32: RELATION OF INITIAL KEYING COST TO ACCURACY



PRICE FOR KEYBOARDING ($SB) AS FUNCTION OF
EDITING/PROOFING COSTS & ERROR RATE

K = total unit cost
(constant)

FIGURE 33: ACCEPTABILITY IN TERMS OF ACCURACY AND COST FOR THREE PRICE QUOTATIONS FOR KEYING



REFERENCES

1. Avram, Henriette D., John F. Knapp, and Lucia J. Rather. The MARC II Format: A Communications Format for Bibliographic Data. Washington, D.C. Information Systems Office, Library of Congress, 1968. 167 pp.

2. Becker, Joseph and Robert M. Hayes. Information Storage and Retrieval: Tools, Elements, Theories. New York, Wiley, 1963. 448 pp.

3. Bregzis, Ritvars. "Query Language for the Reactive Catalogue." In: Tonik, Albert B., ed. Information Retrieval: the User's Viewpoint - An Aid to Design. Philadelphia, International Information, Inc., 1967. pp. 77-91. (Fourth Annual National Colloquium on Information Retrieval, May 3-4, 1967.)

4. Cartwright, Kelley L. and R.M. Shoffner. Catalogs in Book Form: A Research Study of their Implications for the California State Library and the California Union Catalog, with a Design for their Implementation. Berkeley, Institute of Library Research, University of California, 1967. various pagings.

5. Cox, N.S.M. and J.D. Dews. "The Newcastle File Handling System." In: Cox, Nigel S.M. and M.W. Grose, eds. Organization and Handling of Bibliographic Records by Computer. Hamden, Conn., Archon Books, 1967. pp. 1-21.

6. Cunningham, Jay L. Instruction Manual for Editorial Preparation of Catalog Source Data. Preliminary Edition. Berkeley, Institute of Library Research, University of California, 1968. 172 p.

7. Jackson, Michael. "Mnemonics." Datamation, v. 13 (Apr. 1967), pp. 26-28.

8. Mathematical Reviews. (Am. Math. Soc., Lancaster, Pa.) v. 30 (1965), p. 1207.

9. Palmer, Foster M. "Conversion of Existing Records in Large Libraries; with Special Reference to the Widener Library Shelflist." In: Harrison, John and Peter Laslett, eds. The Brasenose Conference on the Automation of Libraries. Held at Oxford, Eng., 30 June - 3 July 1966. London, Mansell, 1967. pp. 57-83.

REFERENCES (Cont.)

10. Payne, Charles T. "Tagging Codes." Chicago, University of Chicago Library, Feb. 1967. (unpublished report) various pagings.

11. Vaughan, Delores K. "Effectiveness of Book-Memory Data for Conventional Catalog Retrieval." In: Chicago. University. Graduate Library School. Requirements Study for Future Catalogs; Progress Report No. 2. Chicago, Mar. 1968. (NSF Grant GN 432) p. 53.

APPENDIX I

AN ALGORITHM FOR NOISY MATCHES IN CATALOG SEARCHING

By

James L. Dolby
R & D Consultants Company
Los Altos, California

AN ALGORITHM FOR NOISY MATCHES IN CATALOG SEARCHING*

By

James L. Dolby
R & D Consultants Company
Los Altos, California

A. INTRODUCTION

A viable on-line search system cannot reasonably assume that each user will invariably provide the proper input information without error. Human beings not only make errors, but they also expect their correspondents, be they human or mechanistic, to be able to cope with these errors, at least at some reasonable error-rate level. Many of the difficulties in implementing computer systems in many areas of human activity stem from failure to recognize, and plan for, routine acceptance of errors in the systems. Indeed, computing did not become the widespread activity it is now until the so-called higher-level languages came into being. Although it is customary to think of higher-level languages as being "more English-like," the height of their level is better measured by the brevity with which various jobs can be expressed (for brevity tends to reduce errors) and the degree of sophistication of their automatic error detection and correction procedures.

The processing of catalog information for the purposes of exposing and retrieving information presents at least two major areas for research in automatic error detection and correction. At the first stage, the data bank must be created, updated and maintained. Methods for dealing with input errors at this level have been derived by a number of groups and it seems reasonable to assert that something in the order of 60 per cent of the input errors can be detected automatically (1,2,3). With the possibility of human proofreading and error detection through actual use, it is reasonable to expect a mature data base to have a very low over-all error rate.

At the second stage, however, when a user approaches the data base through a terminal or other on-line device, the errors will be of a recurring nature: each user will generate his own error set and though experience will tend to minimize the error rate for a particular user, there will be an essentially irreducible minimum error rate even for an experienced user. And if the system is to attract users other than professional interrogators, it must respond intelligently at this minimal error level.

---

*This research was carried out for the Institute of Library Research, University of California, under the sponsorship of the Office of Education, Research Grant No. OEG-1-7-071083-5068.

In this paper we consider certain problems associated with making "noisy matches" in catalog searches. Because preliminary information indicates that the most likely source of input errors is in the keyboarding of proper names, the main emphasis of the paper will be on the problem of algorithmically compressing proper names in such a way as to identify similar names (and likely misspellings) without over-identifying the list of possible authors.

B. THE STRUCTURE OF EXISTING NAME-COMPRESSION ALGORITHMS

The problem of providing equivalence classes of proper names is hardly new. Library catalogs, telephone directories and other major data bases have made use of "see-also"-type references for many years. Some years ago Remington-Rand derived an alphanumeric name compression algorithm, SOUNDEX, that could be applied either by hand or by machine for such purposes (4). Perhaps the most widely used on-line retrieval system presently in existence, the airline reservation system (such as SABRE), makes use of such an algorithm (5). The closely related problem of compressing English words (either to establish noisy matches, eliminate misspelled words, or simply to achieve data bank compression) has also received some attention (see, for example, (6) and (7) and (8)).

Although the English word structure differs from proper-name structure in some important respects (e.g., the existence of suffixes), three of the algorithms are constructed by giving varying degrees of attention to the following five areas of word structure:

1. The character in word initial position
2. The character set: (A,E,I,O,U,Y,H,W)
3. Doubled characters (e.g., tt)
4. Transformation of consonants (i.e., all alphabetic characters other than those in 2 above) into equivalence classes.
5. Truncation of the residual character string.

The word-initial character receives varying attention. SOUNDEX places the initial consonant in the initial position of the compressed form and then transforms all other consonants into equivalence classes with numeric titles. SABRE maintains the word-initial character even if it is a vowel. In the Armour Research Foundation scheme (ARF), the word-initial character is also retained as is.

Both SOUNDEX and SABRE eliminate all characters in the set 2 above. The ARF scheme retains all characters in shorter words and deletes vowels only, to reduce the compressed form to four characters, deleting the U after Q, the second vowel in a vowel string, and then all remaining vowels.

All three systems delete the second letter of a double-letter string. SABRE goes a step further and deletes the second letter of a double-letter string occurring after the vowels have been deleted. Thus, the second R of BEARER would be deleted.

SOUNDEX maps the 18 consonants into 6 equivalence classes as follows:

1. B,F,P,V
2. C,G,J,K,Q,S,X,Z
3. D,T
4. L
5. M,N
6. R

SABRE and ARF do not perform any transformations on these 18 consonants.

Finally, all three systems truncate the remaining string of characters to four characters. For shorter forms, padding in the form of zeros (SOUNDEX), blanks (SABRE), or hyphens (ARF) is added so that all codes are precisely four characters long.

Variable-length coding schemes have been considered but generally rejected for implementation on major systems because of the attendant difficulties of programming and the fact that code compression is enhanced by fixed-length codes where no interword space is necessary. Although fixed-length schemes of length greater than four have been considered, no definitive data appears to be available as to the ability to discriminate by introduction of more characters in the compressed code. The SABRE system does add a fifth character but makes use of the person's first initial for added discrimination.

Tukey (9) has constructed a personal author code for his citation indexing and permuted title studies on an extensive corpus of the statistical literature. In this situation the author code is a semi-mnemonic code in a tag form to assist the user in identification rather than as a basic entry point. However, Tukey does note that in his corpus a three-character code of the surname plus two initials is superior to a five-character surname code for purposes of unique identification.

C. MEASURING ALGORITHMIC PERFORMANCE

One of the main problems in constructing linguistic algorithms is to decide on appropriate measures of performance and to obtain data bases for implementing such measures. In this case it is clear that certain improvements in existing algorithms can be made - particularly by using more sophisticated transformation rules for the consonants - and that the problems of implementing such changes are not so great in today's context

3

as they were when the systems noted above were originally derived. Improvements in processing speeds and programming languages, however, do not remove the need for keeping "linguistic frills" to a minimum.

Ideally, it would be desirable to have a list of common errors in keyboarding names as a test basis for any proposed algorithms. Unfortunately, no such list of sufficient size appears to be available. Lacking this, one can speculate that certain formal properties of the predictability of language might be useful in deriving an algorithm. At the English word level, some effort has been made to exploit measures of entropy as developed by Shannon in this direction (6,7). However, there is good reason to question whether entropy, at least when measured in the usual way, is strongly correlated with actually occurring errors (10).

As an alternative, one can study existing lists of personal-name equivalence classes to derive such algorithms and then test the algorithm against such classes, measuring both the degree of over-identification and the degree of under-identification. Clearly, such tests will carry more weight if they are carried out under economic forcing conditions where weaknesses in the test set will lead to real and measurable expense to the organization publishing the list. The SABRE system operates under strong economic forcing conditions in the sense that airline passengers frequently have a number of competitive alternatives available to them and lost reservations can cause them sufficient inconvenience for them to consider these alternatives. However, the main application of the SABRE system is to rather small groups of persons (at least when compared to the number of personal authors in a typical library catalog) so that errors of over-identification are essentially trivial in cost to the airlines.

A readily available source of see-also-type equivalence classes of proper names is given in the telephone directory system. Here, the economic forcing system is not so strong as in the airline situation, but it is measurable in that failure to provide an adequate list will lead to increased user dependence on the Information Operator - with consequent increased cost to the telephone company. As a test on the feasibility of using such a set of equivalence classes, the 451 classes found in the Palo Alto-Los Altos (California) telephone directory were copied out by hand and used in deriving and testing the algorithm given in the next section and the SOUNDEX algorithm.

There remains the question of deciding what is to constitute proper agreement between any algorithm and the set of equivalence classes chosen as a data base. At the grossest level it seems reasonable to argue that over-identification is less serious than under-identification. False drops only tend to clog the line.

Lost reference points, on the other hand, lead to lost information. Investigation of other applications of linguistic algorithms, such as algorithms to hyphenate words, identify semantically similar words through cutting off of suffixes, and so forth, indicates that it is usually possible to reduce crucial error (in this case under-identification) to something under five percent, while preserving something in the order of 80 percent of the original distinctions (or efficiency) of the system. Efforts to improve materially on the "five-and-eighty" rule generally lead to solutions involving larger context and/or extensive exception dictionaries. In this study we shall aim our efforts at achieving a "five-and-eighty" solution.

D. A VARIABLE-LENGTH NAME-COMPRESSION SCHEME

In light of the fact that no definitive information is available on the problems of truncating errors in name-compression algorithms, it is convenient to break the problem into two pieces. First, we shall derive a variable-length algorithm of the required accuracy and efficiency and then we shall determine the errors induced by truncation.

After studying the set of equivalence classes given in the Palo Alto-Los Altos telephone directory, it was fairly clear that with minor modifications of the basic five steps used in the other algorithms noted above, it would not be too difficult to provide a reasonably accurate match without requiring too much over-identification. The main modifications made consisted of maintaining the position of the first vowel and using local context to make transformations on the consonants. The algorithm is given below.

A Spelling Equivalent Abbreviation Algorithm For Personal Names

1. Transform: McG to Mk, Mag to Mk, Mac to Mk, Mc to Mk.
2. Working from the right, recursively delete the second letter from each of the following letter pairs: dt, ld, nd, nt, rc, rd, rt, sc, sk, st.
3. Transform: x to ks, ce to se, ci to si, cy to sy. Consonant-ch to consonant-sh. All other occurrences of c to k, z to s, wr to r, dg to g, qu to k, t to d, ph to f (after the first letter).
4. Delete all consonants other than l, n, and r which precede the letter k (after the first letter).
5. Delete one letter from any doubled consonant.
6. Transform pf# to p#, #pf to #f, vowel-gh# to vowel-f#, consonant-gh# to consonant-g#, and delete all other occurrences of gh. (# is the word-beginning and word-ending marker.)
7. Replace the first vowel in the name by the symbol "*".
8. Delete all remaining vowels.
9. Delete all occurrences of w or h after the first letter in the word.

The vowels are taken to be (A,E,I,O,U,Y).

The algorithm splits 22 (4.9 percent) of the 451 equivalence classes given by the phone directory. On the other hand, the algorithm provides 349 distinct classes (not counting those classes that were broken off in error) or 77.4 percent of the 451 classes in the telephone directory data base. Thus, we have achieved a reasonable approximation to the "five-and-eighty" performance found in other linguistic problem areas.

To give a proper appreciation of the nature of these under-identification errors, they are discussed below individually.

1. The name Bryer is put in the same equivalence class with a variety of spellings of the name Bear. The algorithm fails to make this identification.
2. Blagburn is not equated to Blackburn.
3. The name Davison is equated to Davidson in its various forms. The algorithm fails to make this identification and this appears to be one of a modest class of difficulties that occur prior to the -son, -sen names.
4. The class of names Dickinson, Dickerson, Dickison, and Dickenson are all equated by the directory but kept separate except for the two forms of Dickinson by the algorithm.
5. The name Holm is not equated with the name Home.
6. The name Holmes is not equated with the name Homes.
7. The algorithm fails to equate Jaeger with the various forms of Yaeger.
8. The algorithm fails to equate Lamb with Lamn.
9. The algorithm incorrectly assumes that the final gh of Leigh should be treated as an f. Treating final gh either as a null sound or an f leads to about the same number of errors in either direction.
10. The algorithm fails on the pairing Leicester and Lester. The difficulty is an intervening vowel.
11. The algorithm fails to equate the various forms of Lindsay with the forms of Lindsley.
12. The algorithm fails to equate the various forms of McLaughlin with McLachlan.
13. The algorithm fails to equate McCullogh with McCullah. This is again the final gh problem.
14. The algorithm fails to equate McCue with McHugh (again the final gh problem).
15. The algorithm fails to equate Moretton with Morton. This is an intervening vowel problem.
16. The algorithm fails to equate Rauch with Roush.
17. The algorithm fails to equate Robinson with Robison (another -son type problem).
18. The algorithm incorrectly assumes that the interior ph of Shepherd is an f.
19. The algorithm fails to equate Speer with Speier.

20. The algorithm fails to equate Stevens with Stephens.
21. Similarly for Stevenson and Stephenson.
22. The algorithm fails to equate the various forms of the word Thompson (an -son problem).

Several of the errors noted above are questionable, at least in the sense of questioning whether the telephone directory is following its own procedures with complete rigor. Setting these aside, the primary errors occur with the final gh, the words ending in -son, and the words with the extraneous interior vowels. Each of these problems can be resolved to any desired degree of accuracy, but only at the expense of noticeable increases in the degree of complexity of the algorithm.

E. THE TRUNCATION PROBLEM

Simple truncation does not introduce errors of under-identification; it can only lead to further over-identification. Examination of the results of applying the algorithm to the telephone directory data base shows that no new over-identification is introduced if the compressed codes are all reduced to the left most seven characters. Further truncation leads to the following short table:

Code Length	Cumulative Over-Identification Losses
7	0
6	1
5	6
4	45

Thus there is a strong argument for maintaining at least five characters in the compressed code.

However, there is no real need for restricting ourselves to simple truncation. Following the procedures used in the ARF system, we can obtain further truncation by selectively removing some of the remaining characters. The natural candidate for such removal is the vowel marker. If the vowel marker is removed from all the five character codes, only six more over-identification errors are introduced. Removal of the vowel markers from all of the codes would have introduced 17 more errors of over-identification. Thus we see that the utility of the vowel marker is in the short codes.

This in turn suggests that introduction of a second vowel marker in the very short codes may have some utility. This is indeed the case. If we generalize the notion of the vowel marker to that of marking the position of a vowel-string (i.e., a string of consecutive vowels), where for these purposes a vowel is any of the characters (A,E,I,O,U,Y,H,W), and maintain these markers as "padding" in the very short words, 18 errors of over-identification are eliminated at the cost of two new errors of under-identification. In this way we derive the following modification to the variable length algorithm:

1. Mark the position of each of the first two vowel strings with an "*", if there is more than one vowel.
2. Truncate to six characters.
3. If the six-character code has two vowel markers, remove the right hand vowel marker. Otherwise, truncate the sixth character.
4. If the resulting five-character code has a vowel marker, remove it. Otherwise remove the fifth character.
5. For all codes having less than four characters in the variable-length form, pad to four characters by adding blanks to the right.

Measured against the telephone directory data base, this fixed length compression code provides 361 distinct classes (not counting improper class splits as separate classes) or 80 percent of the 451 given classes. Twenty-four (5.3 percent) of the classes are improperly split. By way of comparison, the SOUNDEX system improperly splits 135 classes (30 percent) and provides only 287 distinct classes (not counting improperly split classes) or 63.8 percent of the telephone directory data base.

F. ACKNOWLEDGEMENT

The author would like to thank Ralph M. Shoffner and Kelley L. Cartwright for suggesting the problem and for a number of useful comments on existing systems. Allan J. Humphrey was kind enough to program the variable-length version of the algorithm for test purposes.

G. CORPUS OF NAMES USED FOR ALGORITHM TEST

List of personal-name equivalence classes from the Palo Alto-Los Altos Telephone Directory arranged according to the variable length compression code (with the vowel marker * treated as an A for ordering).

*BL	Abel, Abele, Abell, Able
*BRMS	Abrahams, Abrams
*BRMSN	Abrahamson, Abramson
*D	Eddy, Eddie
*DMNS	Edmonds, Edmunds
*DMNSN	Edmondson, Edmundson
*DMS	Adams, Addems
*GN	Eagan, Egan, Eggen
*GR	Jaeger, Yaeger, Yeager
*KN	Aiken, Aikin, Aitken
*KNS	Adkins, Akins
*KR	Acker, Aker
*KR	Eckard, Eckardt, Eckart, Eckert, Eckhardt
*KS	Oakes, Oaks, Ochs
*LBRD	Albright, Allbright
*LD	Elliot, Elliott
*LN	Allan, Allen, Allyn
*LSN	Ohlsen, Olesen, Olsen, Olson, Olsson
*LVR	Oliveira, Olivera, Olivero
*MS	Ames, Eames
*NGL	Engel, Engle, Ingle
*NL	O'Neal, O'Neil, O'Neill
*NRS	Andrews, Andrus
*NRSN	Andersen, Anderson, Andreasen
*NS	Ennis, Enos
*RKSN	Erichsen, Erickson, Ericson, Ericsson, Eriksen
*RL	Earley, Early
*RN	Erwin, Irwin
*RNS	Aarons, Ahrends, Ahrens, Arens, Arentz, Arons
*RS	Ayers, Ayres
*RVN	Ervin, Ervine, Irvin, Irvine
*RVNG	Erving, Irving
*SBRN	Osborn, Osborne, Osbourne, Osburn

Note: Names whose compressed codes do not match the one given in the first column (and hence represent weaknesses in the algorithm and/or the directory groupings) are underlined.

Note: A small number of directory entries that do not bear on the immediate problem have been deleted from the list: Bell's see also Bells; Co-op see also Co-operative; Palo Alto Clinic see also Palo Alto Medical Clinic; St. see also Saint; etc.

B*D	Beatie, Beattie, Beatty, Beaty, Beedie
B*DS	Betts, Betz
B*KMN	Bachman, Bachmann, Backman
B*L	Bailey, Baillie, Bailly, Baily, Bayley
B*L	Beal, Beale, Beall, Biehl
B*L	Belew, Ballou, Bellew
B*L	Buhl, Buell
B*L	Belle, Bell
B*LN	Bolton, Boulton
B*M	Baum, Bohm, Bohme
B*MN	Bauman, Bowman
B*N	Bain, Bane, Bayne
B*ND	Bennet, Bennett
B*R	Baer, Bahr, Baier, Bair, Bare, Bear, Beare, Behr, Beier, Bier, Bryer
B*R	Barry, Beare, Beery, Berry
B*R	Bauer, Baur, Bower
B*R	Bird, Burd, Byrd
B*RBR	Barbour, Barber
B*RG	Berg, Bergh, Burge
B*RGR	Berger, Burger
B*RK	Boerke, Birk, Bourke, Burk, Burke
B*RN	Burn, Byrne
B*RNR	Bernard, Bernhard, Bernhardt, Bernhart
B*RNS	Berns, Birns, Burns, Byrns, Byrnes
B*RNSN	Bernstein, Bornstein
B*RS	Bertsch, Birch, Burch
BL*KBRN	Blackburn, Blagburn
BL*M	Blom, Bloom, Bluhm, Blum, Blume
BR*D	Brode, Brodie, Brody
BR*N	Braun, Brown, Browne
BR*N	Brand, Brandt, Brant
D*DS	Dietz, Ditz
D*F	Duffie, Duffy
D*GN	Dougan, Dugan, Duggan
D*K	Dickey, Dicke
D*KNSN	Dickenson, Dickerson, Dickinson, Dickison
D*KSN	Dickson, Dixon, Dixson
D*L	Dailey, Daily, Daley, Daly
D*L	Dahl, Dahle, Dall, Doll
D*L	Deahl, Deal, Diehl
D*MN	Diamond, Dimond, Dymond
D*N	Dean, Deane, Deen
D*N	Denney, Denny
D*N	Donahoo, Donahue, Donoho, Donohoe, Donohoo, Donohue, Dunnahoo
D*N	Downey, Downie
D*N	Dunn, Dunne
D*NL	Donley, Donnelley, Donnelly
D*R	Daugherty, Doherty, Dougherty
D*R	Dyar, Dyer
D*RM	Derham, Durham
D*VDSN	Davidsen, Davidson, Davison

D*VS	Davies, Davis
DR*SL	Driscoll, Driskell
F*	Fay, Fahay, Fahey
F*FR	Fifer, Pfeffer, Pfeiffer
F*GN	Fagon, Feigan, Fegan
F*L	Feil, Pfeil
F*L	Feld, Feldt, Felt
F*LKNR	Faulkner, Falconer
F*LPS	Philips, Phillips
F*NGN	Finnegan, Finnigan
F*NL	Finlay, Finley
F*RL	Farrell, Ferrell
F*RR	Ferrara, Ferreira, Ferriera
F*RR	Foerster, Forester, Forrester, Forster
F*RS	Forrest, Forest
F*RS	Faris, Farriss, Ferris, Ferriss
F*RS	First, Fuerst, Furst
F*SR	Fischer, Fisher
FL*N	Flinn, Flynn
FL*NGN	Flanagan, Flanigan, Flannigan
FR*	Frei, Frey, Fry, Frye
FR*DMN	Freedman, Friedman
FR*DRKSN	Frederickson, Frederiksen, Fredrickson, Fredriksson
FR*K	Franck, Frank
FR*NS	France, Frantz, Franz
FR*NS	Frances, Francis
FR*S	Freeze, Freese, Fries
FR*SR	Fraser, Frasier, Frazer, Frazier
G*D	Good, Goode
G*DS	Getz, Goetz, Goetze
G*F	Goff, Gough
G*L	Gold, Goold, Gould
G*LMR	Gilmer, Gilmore, Gilmour
G*LR	Gallagher, Gallaher, Galleher
G*MS	Gomes, Gomez
G*NR	Guenther, Gunther
G*NSLS	Gonzales, Gonzalez
G*NSLVS	Gonsalves, Gonzalves
G*RD	Garratt, Garrett
G*RD	Garrity, Geraghty, Geraty, Gerrity
G*RN	Gorden, Gordohn, Gordon
G*RNR	Gardiner, Gardner, Gartner
G*RR	Garrard, Gerard, Gerrard, Girard
G*S	Gauss, Goss
GR*	Gray, Grey
GR*FD	Griffeth, Griffith
GR*N	Green, Greene
GR*S	Gros, Grose, Gross
H*D	Hyde, Heidt
H*F	Hoff, Hough, Huff
H*FMN	Hoffman, Hoffmann, Hofman, Hofmann, Huffman
H*G	Hoag, Hoge, Hogue

H*GN	Hagan, Hagen
H*K	Hauch, Hauck, Hauk, Hauke
H*KSN	Hutcheson, Hutchison
H*L	Holley, Holly
H*L	Holl, Hall
H*L	Halley, Haley
H*L	Haile, Hale
H*LD	Holiday, Halliday, Holladay, Holliday
H*LG	Helwig, Hellwig
H*LM	Holm, Home
H*LMS	Holmes, Homes
H*LN	Highland, Hyland
H*M	Ham, Hamm
H*MR	Hammar, Hammer
H*N	Hanna, Hannah
H*N	Hahn, Hahne, Hann, Haun
H*NN	Hanan, Hannan, Hannon
H*NRKS	Hendricks, Hendrix, Henriques
H*NRKSN	Hendrickson, Henriksen, Henrikson
H*NS	Heintz, Heinz, Heinze, Hindes, Hinds, Hines, Hinze
H*NS	Haines, Haynes
H*NSN	Henson, Hansen, Hanson, Hanssen, Hansson, Hanszen
H*R	Herd, Heard, Hird, Hurd
H*R	Hart, Hardt, Harte, Heart
H*R	Hare, Hair
H*R	Hardey, Hardie, Hardy
H*RMN	Hartman, Hardmen, Hardmon, Hartmann
H*RMN	Herman, Hermann, Herrmann
H*RMN	Harman, Harmon
H*RN	Heron, Herrin, Herron
H*RN	Hardin, Harden
H*RN	Horn, Horne
H*RNGDN	Herrington, Harrington
H*S	Haas, Haase, Hasse
H*S	Howes, House, Howse
H*S	Hays, Hayes
H*SN	Houston, Huston
H*VR	Hoover, Hover
J*	Jew, Jue
J*FR	Jeffery, Jeffrey
J*FRS	Jefferies, Jefferis, Jefferys, Jeffreys
J*KB	Jacobi, Jacoby
J*KBSN	Jacobsen, Jacobson, Jakobsen
J*KS	Jacques, Jacks, Jaques
J*L	Jewell, Juhl
J*MS	Jaimes, James
J*MSN	Jameson, Jamieson, Jamison
J*NSN	Jahnsen, Jansen, Jansohn, Janssen, Jansson, Janzen, Jensen, Jenson
J*S	Joice, Joyce

K*	Kay, Kaye
K*F	Coffee, Coffey
K*FMN	Coffman, Kauffman, Kaufman, Kaufmann
K*K	Cook, Cooke, Koch, Koche
K*L	Cole, Kohl, Koll
K*L	Kelley, Kelly
K*LMN	Coleman, Colman
K*LR	Koehler, Koeller, Kohler,Koller
K*MBRLN	Chamberlain, Chamberlin
K*MBS	Combs, Coombes, Coombs
K*MP	Camp, Kampe, Kampf
K*MPS	Campos, Campus
K*N	Cahn, Conn, Kahn
K*N	Cahen, Cain, Caine, Cane, Kain, Kane
K*N	Chin, Chinn
K*N	Chaney, Cheney
K*N	Coen, Cohan, Cohen, Cohn, Cone, Koehn, Kohn
K*N	Coon, Kuhn, Kuhne
K*N	Kenney, Kenny, Kinney
K*NL	Conley, Conly, Connelly, Connolly
K*NR	Conner, Connor
K*NS	Coons, Koontz, Kuhns, Kuns, Kuntz, Kunz
K*P	Coop, Co-op, Coope, Coupe, Koop,
K*PL	Chapel, Chapell, Chappel, Chappell, Chappelle, Chapple
K*R	Carrie, Carey, Cary
K*R	Corey, Cory
K*R	Carr, Kar, Karr
K*R	Kurtz, Kurz
K*R	Kehr, Ker, Kerr
K*RD	Cartwright, Cortright
K*RLN	Carleton, Carlton
K*RN	Carney, Cerney, Kearney
K*RSNR	Kirschner, Kirchner
K*S	Chace, Chase
K*S	Cass, Kass
K*S	Kees, Keyes, Keys
K*SL	Cassel, Cassell, Castle
K*SLR	Kesler, Kessler, Kestler
K*SR	Kaiser, Kayser, Keizer, Keyser, Kieser, Kiser, Kizer
KL*N	Cline, Klein, Kleine, Kline
KL*RK	Clark, Clarke
KL*SN	Claussen, Clausen, Clawson, Closson
KR*	Crow, Crowe
KR*GR	Krieger, Kroeger, Krueger, Kruger
KR*MR	Creamer, Cramer, Kraemer, Kramer, Kremer
KR*N	Craine, Crane
KR*S	Christie, Christy, Kristee
KR*S	Crouss, Kraus, Krausch, Krause, Krouse
KR*S	Cross, Krost
KR*S	Crews, Cruz, Kruse
KR*SNSN	Christensen, Christiansen, Christianson

L*	Loe, Loewe, Low, Lowe
L*	Lea, Lee, Leigh
L*D	Lloyd, Loyd
L*DL	Litle, Littell, Little, Lytle
L*DRMN	Ledterman, Letterman
L*K	Leach, Leech, Leitch
L*KS	Lucas, Lukas
L*LN	Laughlin, Loughlin
L*LR	Lawler, Lawlor
L*MB	Lamb, Lamm
L*MN	Lemen, Lemmon, Lemon
L*MN	Layman, Lehman, Lehmann
L*N	Lind, Lynd, Lynde
L*N	Lion, Lyon
L*N	Lin, Linn, Lynn, Lynne
L*N	Lain, Laine, Laing, Lane, Layne
L*NG	Lang, Lange
L*NN	London, Lundin
L*NS	Lindsay, Lindsey, Lindsley, Linsley
L*R	Lawry, Lowery, Lowrey, Lowry
L*RNS	Lawrence, Lowrance
L*RNS	Laurence, Lawrance, Lawrence, Lorence, Lorenz
L*RSN	Larsen, Larson
L*S	Lewis, Louis, Luis, Luiz
L*S	Lacey, Lacy
L*SR	Leicester, Lester
L*V	Levey, Levi, Levy
L*VD	Leavett, Leavitt, Levit
L*VL	Lavell, Lavelle, Leavelle, Loveall, Lovell
L*VN	Lavin, Levin, Levine
M*D	Mead, Meade
M*DN	Morretton, Morton
M*DS	Mathews, Matthews
M*DSN	Madison, Madsen, Matson, Matteson, Mattison, Mattson
M*KL	Michael, Michel
M*KM	Meacham, Mechem
M*KS	Marques, Marquez, Marquis, Marquiss
M*KS	Marcks, Marks, Marx
M*LN	Maloney, Moloney, Molony
M*LN	Mullan, Mullen, Mullin
M*LR	Mallery, Mallory
M*LR	Moeller, Moller, Mueller, Muller
M*LR	Millar, Miller
M*LS	Miles, Myles
M*N	Mahan, Mann
M*NR	Miner, Minor
M*NR	Monroe, Munro
M*NSN	Monson, Munson
M*R	Murray, Murrey
M*R	Maher, Maier, Mayer
M*R	Mohr, Moor, Moore
M*R	Meyers, Myers
M*R	Meier, Meyer, Mieir, Myhre

M*RF	Murphey, Murphy
M*RL	Merrell, Merrill
M*RN	Marten, Martin, Martine, Martyn
M*RS	Meyers, Myers
M*RS	Maurice, Morris, Morse
MK*	McCoy, McCaughey
MK*	Magee, McGee, McGehee, McGhie
MK*	Mackey, MacKay, Mackie, McKay
MK*	McCue, McHugh
MK*L	Magill, McGill
MK*LF	McCollough, McCullah, McCullough
MK*LM	McCallum, McCollum, McColm
MK*N	McKenney, McKinney
MK*NR	MacIntyre, McEntire, McIntire, McIntyre
MK*NS	MacKenzie, McKenzie
MK*NS	Maginnis, McGinnis, McGuinness, McInnes, McInnis
MK*R	Maguire, McGuire
MK*R	McCarthy, McCarty
MKD*NL	MacDonald, McDonald, McDonnell
MKF*RLN	MacFarland, MacFarlane, McFarland, McFarlane
MKF*RSN	MacPherson, McPherson
MKL*D	MacLeod, McCloud, McLeod
MKL*KLN	MacLachlan, Maclachlin, McLachlan, McLaughlin, McLoughlin
MKL*LN	McClellan, McClelland, McLellan
MKL*N	McClain, McClaine, McLain, McLane
MKL*N	MacLean, McClean, McLean
MKL*S	McCloskey, McClosky, McCluskey
MKM*LN	MacMillan, McMillan, McMillin
MKN*L	MacNeal, McNeal, McNeil, McNeill
MKR*D	Magrath, McGrath
N*KL	Nichol, Nicholl, Nickel, Nickle, Nicol, Nicoll
N*KLS	Nicholls, Nichols, Nickels, Nickles, Nicols
N*KLS	Nicholas, Nicolas
N*KLSN	Nicholsen, Nicholson, Nicolaisen, Nicolson
N*KSN	Nickson, Nixon
N*L	Neal, Neale, Neall, Neel, Neil, Neill
N*LSN	Neilsen, Neilson, Nelsen, Nelson, Nielsen, Nielson, Nilson, Nilssen, Nilsson
N*MN	Neumann, Newman
N*RS	Norris, Nourse
N*SBD	Nesbit, Nesbitt, Nisbet
P*D	Pettee, Petty
P*DRSN	Peterson, Pederson, Pedersen, Petersen, Petterson
P*G	Page, Paige
P*LK	Polak, Pollack, Pollak, Pollock
P*LSN	Polson, Paulsen, Paulson, Poulsen, Poulsson
P*N	Paine, Payn, Payne
P*R	Parry, Perry
P*R	Parr, Paar
P*RK	Park, Parke
P*RKS	Parks, Parkes

P*RS	Pierce, Pearce, Peirce, Piers
P*RS	Parish, Parrish
P*RS	Paris, Parris
P*RSN	Pierson, Pearson, Pehrson, Peirson
PR*KR	Prichard, Pritchard
PR*NS	Prince, Prinz
PR*R	Prior, Pryor
R*	Roe, Rowe
R*	Rae, Ray, Raye, Rea, Rey, Wray
R*BNSN	Robinson, Robison
R*D	Rothe, Roth
R*D	Rudd, Rood, Rude
R*D	Reed, Read, Reade, Reid
R*DR	Rider, Ryder
R*DS	Rhoades, Rhoads, Rhodes
R*GN	Regan, Ragon, Reagan
R*GR	Rodgers, Rogers
R*K	Richey, Ritchey, Ritchie
R*K	Reich, Reiche
R*KR	Reichardt, Richert, Rickard
R*L	Reilley, Reilly, Reilli, Riley
R*MNGTN	Remington, Rimington
R*MR	Reamer, Reimer, Riemer, Rimmer
R*MS	Ramsay, Ramsey
R*N	Rhein, Rhine, Ryan
R*NR	Reinhard, Reinhardt, Reinhart, Rhinehart, Rinehart
R*S	Reas, Reece, Rees, Reese, Reis, Reiss, Ries
R*S	Rauch, Rausch, Roach, Roche, Roush
R*S	Rush, Rusch
R*S	Russ, Rus
R*VS	Reaves, Reeves
S*BR	Seibert, Siebert
S*FL	Schofield, Scofield
S*FN	Stefan, Steffan, Steffen, Stephan, Stephen
S*FNS	Steffens, Stephens, Stevens
S*FNSN	Steffensen, Steffenson, Stephenson, Stevenson
S*FR	Schaefer, Schaeffer, Schafer, Schaffer, Shafer, Shaffer, Sheaffer
S*FR	Stauffer, Stouffer
S*GL	Siegal, Sigal
S*GLR	Sigler, Ziegler
S*K	Schuck, Shuck
S*KS	Sachs, Sacks, Saks, Sax, Saxe
S*L	Seeley, Seely, Seley
S*L	Schell, Shell
S*LR	Schuler, Schuller
S*LS	Schultz, Schultze, Schulz, Schulze, Shults, Shultz
S*LV	Silva, Sylva
S*LVR	Silveira, Silvera, Silveria
S*MKR	Schomaker, Schumacher, Schumaker, Shoemaker, Shumaker

S*MN	Simon, Symon
S*MN	Seaman, Seemann, Semon
S*MRS	Somers, Sommars, Sommers, Summers
S*MS	Simms, Sims
S*N	Stein, Stine
S*N	Sweeney, Sweeny, Sweney
S*NR	Senter, Center
S*NRS	Sanders, Saunders
S*PR	Shepard, Shephard, Shepheard, Shepherd, Sheppard
S*R	Stahr, Star, Starr
S*R	Stewart, Stuart
S*R	Storey, Story
S*R	Saier, Sayre
S*R	Schwartz, Schwarz, Schwarze, Swartz
S*RL	Schirle, Shirley
S*RLNG	Sterling, Stirling
S*RMN	Scheuermann, Schurman, Sherman
S*RN	Stearn, Stern
S*RR	Scherer, Shearer, Sharer, Sherer, Sheerer
S*S	Sousa, Souza
SM*D	Smith, Smyth, Smythe
SM*D	Schmid, Schmidt, Schmit, Schmitt, Smit
SN*DR	Schneider, Schnieder, Snaider, Snider, Snyder
SN*L	Schnell, Snell
SP*LNG	Spalding, Spaulding
SP*R	Spear, Speer, Speirer
SP*R	Spears, Speers
SR*DR	Schroder, Schroeder, Schroeter
SR*DR	Schrader, Shrader
T*D	Tait, Tate
T*MSN	Thomason, Thompson, Thomsen, Thomson, Tomson
T*RL	Terrel, Terrell, Terrill
TR*S	Tracey, Tracy
V*L	Vail, Vaile, Vale
V*L	Valley, Valle
V*R	Vieira, Vierra
W*D	White, Wight
W*DKR	Whitacre, Whitaker, Whiteaker, Whittaker
W*DL	Whiteley, Whitley
W*DMN	Whitman, Wittman
W*DR	Woodard, Woodward
W*DRS	Waters, Watters
W*GNR	Wagener, Waggener, Wagoner, Wagner, Wegner, Waggoner
W*L	Willey, Willi
W*L	Wiley, Wylie
W*L	Wahl, Wall
W*LBR	Wilber, Wilbur
W*LF	Wolf, Wolfe, Wolff, Woolf, Woulfe, Wulf, Wulff
W*LKNS	Wilkens, Wilkins
W*LKS	Wilkes, Wilks
W*LN	Whalen, Whelan
W*LR	Walter, Walther, Wolter

W*LRS	Walters, Walthers, Wolters
W*LS	Wallace, Wallis
W*LS	Welch, Welsh
W*LS	Welles, Wells
W*LSN	Willson, Wilson
W*N	Winn, Wynn, Wynne
W*R	Worth, Wirth
W*R	Ware, Wear, Weir, Wier
W*RL	Wehrle, Wehrlie, Werle, Worley
W*RNR	Warner, Werner
W*S	Weis, Weiss, Wiese, Wise, Wyss
W*SMN	Weismann, Weissman, Weseman, Wiseman, Wismonn, Wissman

REFERENCES

1. Cox, N.S.M. and J.L. Dolby. "Structured Linguistic Data and the Automatic Detection of Errors." *In*: *Advances in Computer Typesetting*. London, Institute of Printing, 1966. pp. 122-125.

2. Cox, N.S.M., J.D. Dews, and J.L. Dolby. *The Computer and the Library*. Hamden, Conn., Archon Press, 1967.

3. Dolby, J.L. "Efficient Automatic Error Detection in Bibliographic Records." R & D Consultants Company Report, April 1968.

4. Becker, Joseph and Robert M. Hayes. *Information Storage and Retrieval*. New York, Wiley, 1963. p. 143.

5. Davidson, Leon. "Retrieval of Misspelled Names in Airlines Passenger Record System." *Communications of the ACM*, v. 5 (1962), pp. 169-171.

6. Blair, C.R. "A Program for Correcting Spelling Errors." *Information & Control*, v. 3 (1960), pp. 60-67.

7. Schwartz, E.S. "An Adaptive Information Transmission System Employing Minimum Redundancy Word Codes." Armour Research Foundation Report, April 1962. (AD 274-135).

8. Bourne, C.P. and D. Ford. "A Study of Methods for Systematically Abbreviating English Words and Names." *Journal of the ACM*, v. 8 (1961), pp. 538-552.

9. Tukey, J.W. "A Tagging System for Journal Articles and Other Citable Items: a Status Report." Statistical Techniques Research Group, Princeton University, 1963.

10. Resnikoff, A. and J.L. Dolby. "A Proposal to Construct a Linguistic and Statistical Programming System." R & D Consultants Company, 1967.

APPENDIX II

USER'S GUIDE TO THE TERMINAL MONITOR SYSTEM (TMS)

By

William D. Schieber
Institute of Library Research
University of California
Berkeley, California

USER'S GUIDE TO THE TERMINAL MONITOR SYSTEM (TMS)

By

William D. Schieber
Institute of Library Research
University of California
Berkeley, California

A. INTRODUCTION

The Terminal Monitor System (TMS) provides on-line terminal access to the Computer Center's IBM 360/40. All files are maintained on ILR's private disk facilities and are not accessible by other 360 users. The system performs five general functions:

1. Text entry: the establishment of new files in which the records can later be processed randomly.
2. File search: retrieval and display of records within an existing file.
3. Text editing: addition, replacement, and deletion of character strings and individual records, within an existing file.
4. Compilation of source programs: conversion to executable instructions from source language entered in same fashion as text.
5. Interface to special user-written routines: ability for terminal user to load and execute special-purpose programs.

B. GENERAL DESCRIPTION OF TMS

TMS has a time sharing design which allows multiple terminals to operate at the same time. Like any conversational system it allows the user to carry on a dialog with the computer, and will wait for the user to enter his response before continuing the processing. Descriptions of the formats of monitor messages and user responses to these messages are described in the following sections. One section is included for each of the first three of the processing modules. The last two functions will be described in a subsequent edition of this guide.

C. 2740 OPERATION

The IBM 2740 terminal is, in most respects, similar to a normal typewriter. In order to use it as a computer terminal, the switch on the right marked 'COM' and 'LCL' must be set to 'COM', and the ON-OFF switch set to 'ON'. At this point the standby light (marked 'S') should be on. Messages sent to the terminal by the monitor will cause the receive ('R') light to go on during the transmission. When the terminal is requested to send text (Transmit) the user must press the 'Bid' key (which causes the transmit ('T') light to go on). At this point he may type the message he wants to send. The message must be ended with the 'EOT' character. This signals the monitor that the terminal has completed transmission, so that it may process the incoming text.

D. DETAILED DESCRIPTION OF UTILITY PROCESSORS

There are six different utility processors which the terminal user may summon. However, before calling any of these for the first time he will be asked by the monitor to log in. The format is currently as follows:

The monitor sends:

```
TERMINAL READY--PLEASE LOG IN
```

Terminal user responds with his name:

```
doe, john
```

Following this, the monitor will respond:

```
THANK YOU--TERMINAL CLEAR
```

At this point the terminal is ready to enter any processor existing in the ILR processor library. Processors currently operational are:

1. *Text Processor.* The text processor is used to create a new file. Records so entered can be retrieved later by Key. The keys generated by the processor, have numeric value. To enter this processor, type in:

```
text
```

TMS responds:

```
TEXT PROCESSOR--TYPE FILE NAME
```

At this point, type the name by which you wish the file to be known. If you are typing a source program, the file name *must* be the name of the entry point (in assembler: the CSECT name; in PL/1: the PROCEDURE name). The file name may be up to 8 characters in length. The format of it is:

```
(name of file)
```

If this name has not been used before, you will receive space for the new file on the disk; before beginning text entry, you will be asked to indicate the treatment of lower case characters in creating the file:

```
SPECIFY CASE
```

If you wish lower case letters to be translated into upper case, (you *must* if you are typing a program), type:

```
l/u
```

(indicating lower to upper case translation)

If you wish lower case characters to maintain their values, type:

`1/1` (for lower to lower case translation)

If you had specified '1/u' case, TMS would respond with:

`L/U CASE--BEGIN TEXT ENTRY`

and would go on to the next line to type the first key:

`0010`

You may now enter the first record. It must not exceed 72 characters and must end with an "EOT". Do not enter a carriage return at the end of the line: TMS will do this before it types the next record number.

To close the file when you have finished entering text, type '+++' in position 1. TMS responds:

`EXIT TEXT PROCESSOR`

2. Search Processor. The search routines are used to display one or more records from an existing file. To call the processor, type:

`srch`

TMS will respond:

`SEARCH ROUTINES--SELECT FILE`

User responds by typing the file name:

`(file name)`

TMS, after locating the file, will respond:

`TYPE SEARCH REQUEST`

Two facilities for search are available. One is display of one or more records, where the records are identified by the exact full key. If one specific record is desired, type:

`display` `(key)`

If you want to display a group of records type:

`display` `(key 1)` `to` `(key 2)`

where the value of key$_1$ is less than key$_2$, and key$_1$ is an actual key in the file. Both values must be of full length.

The second type of search enables the user to scan the file using abbreviated key prefixes, or portions of the key. In this type of search all records of the given key class will be presented. For example, if a given file contains records having keys of 0004, 0010, 0015, and 0020, and a scan on key class 001 is requested, only records whose leftmost three key characters match the three-character scan value will be presented; here, records 0010 and 0015. The format for this request is:

`scan` `(key class)`

After a search request is entered, records are presented sequentially until the request has been satisfied, or until the end of file is reached. However, to enable the user to stop printing of the file, a checkpoint is entered following printing of the tenth record. If you want to continue display, type a carriage return. If you want to stop processing the request, type three plus signs '+++'.

When the request is satisfied, or when it has been interrupted at a checkpoint TMS will type:

```
TYPE SEARCH REQUEST
```

at which time the user may enter another request or leave the search mode. To exit, type '+++'. TMS then responds:

```
EXIT SEARCH PROCESSOR
```

3. Edit Processor. This facility is used to edit existing files. There are three conventional edit functions: replacement, addition, and deletions. These functions may be performed on individual characters, on character strings, on records, or on groups of records within a file.

The edit processor is called by typing:

```
edit
```

TMS responds:

```
EDIT MODE -- SELECT FILE
```

User types in:

```
(file name)
```

TMS replies:

```
SPECIFY CASE
```

To which the user must respond:

`l/u` if he wishes lower case characters to be mapped into upper, or:

`l/l` if he wishes lower to retain their values.

TMS now responds with:

```
PLACE EDIT REQUEST
```

The general format for an edit request is:

```
(code) | + | (key) | + | (character string) | +
```

where 'code' is a two-character code which identifies the edit function, '+' is the tab character, 'key' is the actual key value of a record, and 'character string' represents data to be added to, replaced in, or deleted from the record.

The edit codes are summarized in the table below:

Edit Function	Done On	
	Characters	Records
Replacement	RC	——
Addition	AC	AR
Deletion	DC	DR

Specific edit formats are as follows:

1) REPLACE CHARACTERS - RC.

RC	+	(key)	+	(old string)	+	(new string)	+

where 'old string' is the group of characters which are to be replaced by those presented in the 'new string'.

2) ADD CHARACTERS - AC.

AC	+	(key)	+	(old string)	+	(new string)	+

where 'old string' is a character sequence following which the 'new string' will be inserted.

3) DELETE CHARACTERS - DC.

DC	+	(key)	+	(string)	+

where 'string' is the character sequence to be deleted.

4) ADD RECORD - AR.

AR	+	(key)	+	(new record)

where 'key' is the key of the new record.

5) DELETE RECORD - DR.

DR	+	(key)	+

where key is the key of the record to be deleted.

Following execution of the edit request TMS will indicate completion and invite you to place your next edit request. When you have no more editing to do on the current file you may exit by typing three plus signs. TMS then responds with

```
EDIT MODE EXIT
TERMINAL CLEAR
```

.

APPENDIX III

A DESCRIPTION OF LYRIC, A LANGUAGE FOR REMOTE INSTRUCTION BY COMPUTER

By

Stephen S. Silver
Institute of Library Research
University of California
Los Angeles, California

144/145

A DESCRIPTION OF LYRIC, A LANGUAGE FOR REMOTE INSTRUCTION BY COMPUTER

By

Stephen S. Silver
Institute of Library Research
University of California
Los Angeles, California

A. PURPOSE AND SCOPE

LYRIC (Language for Your Remote Instruction by Computer) was originally intended by its authors, Gloria and Leonard Silvern, for use in computer-assisted instruction (CAI).* The language is, however, so simple and general that it can be used with ease in on-line processor simulation and tightly controlled interactive situations. The instructional strategist (programmer) can selectively display text, and read and analyze user-supplied answers. Branching and jumping based on user-supplied information may be performed. These text display and answer checking functions have been implemented at the Institute of Library Research (Los Angeles) using IBM 2260 display consoles and the System/360 model 75 computing system, controlled by OS/360.

In addition, the full language implementation will give the programmer the ability to set counters and constants, store text, and display this stored information at any time. It will be possible to identify each student uniquely and to record his answers and progress in the instructional program.

The current LYRIC monitor has infinite loop protection. When eighty operations are performed without an intervening write command, a read operation is forced. Typing the word 'end' at the top left of the screen will return control to the time-sharing monitor.

B. LYRIC RELEASE II

1. General Features. Release II of LYRIC will, when completed, have enough data acquisition and manipulation commands to perform most of the error checking functions normally required by batch scanning programs. The goal of this release is to give an information retrieval specialist the ability to write a LYRIC program that will make requests, accept answers, test the answers for acceptability, and modify a file used by other information processing programs.

The other use for LYRIC is for its original intended purpose of computer assisted instruction. Courses can be prepared to give a student enough background to use an information system or for any course of study that lends itself to this technique.

---

*Silvern, Gloria M. and Leonard C. Silvern. "Computer-assisted instruction: specification of attributes for CAI programs and programmers." Association for Computing Machinery. Proceedings, 21st National Conference. Washington, D.C., Thompson, 1966. pp. 57-62.

Specifically, LYRIC will have many of the answer manipulation and storage commands that were lacking in the first ILR implementation. A whole series of inter-program linkage commands will be written to allow LYRIC to control the flow of an information search.

2. Limitation of the Current Release. The major limitation of this release will be its inability to format output files in any precise manner. Output will be in a format similar to OS/360's RECFM=VB. Other limitations will be the techniques for indexing counters. Only addition and subtraction will be allowed at first.

3. Card Format. Each LYRIC statement is one line long and consists of a label, an operation code, and a text or operand field. The label is at column 1, the operation code at 10, and the text at 16.

4. Blocking of Lyric Records. The LYRIC executer will be able to sense and deblock records up to 12 cards per physical record. This block size is controlled by the BLKSIZE sub-parameter in the DCB parameter on the OUTPUT DD card used by the assembler.

5. Implementation Restrictions. The current implementation restrictions are:

1) labels shall be = 8 characters in length;
2) operation codes shall be = 5 characters in length;
3) text or operand fields shall be = 40 characters in length.

All constants shall be S360 halfwords, except string-defining constants which shall be two 1-byte instructions packed into a halfword. One byte will be a length specification and the other will be a location for the start of the operand.

6. Basic Programming Concepts. Executor and assembler optimization will take second place to logic and simplicity of structure. Use of subroutines (BAL type) will be used whenever possible. Table look-up will be used whenever feasible to facilitate debugging and modification of the system. Optimization for a particular system will NOT be performed in this release. Absolute references to a LYRIC record will not be done. Only relative statement numbers will be used. Input/Output will be well defined and in subroutine form to allow use on smaller machines not having a universal character set.

C. THE LYRIC ASSEMBLER

The LYRIC assembler processes LYRIC source input, converting it to executable LYRIC object code. The assembler is divided into two passes. Pass 1 reads in all of the LYRIC source data, modifies an interpretable LYRIC object data set, and generates numerous label tables for use in pass 2. Pass 1 updates and completes the object module produced in pass 2, based on label table information and generates a complete listing of the LYRIC program including any error messages which may have been generated.

1. Pass I. Pass 1 reads in each LYRIC source statement sequentially, producing an object record for each input statement. Before processing each statement, error flags which are analyzed and printed in pass 2, are cleared. The default TRUE and FALSE LYRIC branch locations are set to the default value of the next relative LYRIC location. A running "current LYRIC location" is kept and updated each time a LYRIC source statement is read in. This location value is the basis for most entries in the various label tables.

Explicit labels on source statements are entered into a general label table which saves the literal value of the label and its location value relative to the beginning of the LYRIC data set (using the previously mentioned location pointer). If the label table is full, an error flag is generated within the object LYRIC statement and no entry is made in the label table.

Pass 1 continues with a branch to an operation code analysis routine. This routine is divided into two sections: section 1 searches a list of all known valid op codes. If the op code under analysis is not entered in this table another error flag is set in the object record and processing continues by moving the operation code, text, and label information into the in-core object record, writing it out in the object data set (error flags and all) and looping back to read more LYRIC source statements. If the operation code is found inthis section, control is turned over to the appropriate routine in section 2.

Section 2 is a series of macro-defined subroutines containing all the information necessary to set pointers and otherwise process a LYRIC op code. These subroutines can define the existence of a statement, decide whether or not a LYRIC block starts at this point, decide whether a LYRIC block can end at this statement, define the setting of object constants within the object module, and decide the branching to NEX or PRO. These functions are performed by the use of some standard utility routines which modify label tables and analyze operands.

An indicated branch to NEX or the start of the next LYRIC block is accomplished by setting a global flag indicating that a NEX branch is being attempted, setting a flag in the LYRIC object record indicating that a NEX has been requested and a label resolution is to be performed in pass 2. A system generated number is loaded into the TRUE branch constant of the object record. This system number is the same number that will be forced into the general label table when the next statement that may legally start a LYRIC block is encountered and the NEX global flag is on.

A branch to PRO is resolved in pass 1 by loading the present value of the current step (which is updated each time an op code that starts a step is encountered) into the TRUE location and setting an indicator flag in the record showing that the TRUE pointer has been set.

At present operands are analyzed as either numerical fields or text-character strings. By specifying what you wish to be searched for, subroutines are invoked which search the proper fields on the source statement and which load corresponding constant areas in the object statement. Any error encountered will be flagged in the object statement.

The assembler parses an operand in a number of separate phases for simplicity of coding and debugging.

Phase 1 (STRUCTURE) scans the whole operand field from left to right looking for operand separator delimiters (usually ''') or bracketed by blanks. Starting at the boundaries of the operand subfield, the search starts from the left to the right until either a non-blank or a string delimiter is reached. The scan then searches from right to left for a non-blank or string delimiter. The routine then returns the true length of the searchable part of the string (less the string delimiters, if any) and the starting point of the string (also less string delimiters) relative to the first position of the whole operand field.

A possible difference for the quoted string is for a verbatim test of the answer, starting and ending at the correct boundaries. The unquoted string should be used for logical answers where the operand is a logical word or phrase which is logically but not physically surrounded by blanks. The difference between the two should be interpreted by the executor. The real differences come only at the boundaries of the answer where a non-existent blank overhang should be assumed for non-quoted strings.

2. Pass II. Pass 2 produces a final LYRIC listing and resolves all label references, which results in a fully executable LYRIC object module.

When the last source statement encountered in pass 1 is processed, the object data set is closed, rewound, and opened for update so that the labels can be resolved in place.

Pass 2 now reads the object record as input. As each record is read a subroutine is called which tests the object flags to see if a label resolution must be done. If so, a match is attempted comparing either the label table entries with the explicit label in the object record or the label table's implicit system labels with the system label value saved in the TRUE pointer. Any unresolved labels will print out an error message.

If a branch to the next phrase is required, a subroutine will extract the relative length of the branch from a table generated in pass 1 and add it to the current location, thus generating the relative location of the next phrase. This value is then loaded into the FALSE pointer.

After these operations are finished the completed object record is printed in an expanded format, followed by any error messages that may have been generated. The object record is updated to its final value and the program continues with the next object record.

When the object deck has been exhausted, assembly is terminated.

D. THE LYRIC EXECUTOR

The LYRIC executor is designed to interpret a LYRIC object data set produced by the LYRIC assembler. While the LYRIC assembler is - for most purposes - device independent, the executor is quite machine dependent. The executor must interface between the intent of the LYRIC programmer, the computer operating system, and the particular graphical output terminal being used. The first function of the executor is to try to find and open a LYRIC student record. This record saves status information on a particular student and can be displayed only by the instructor or, under certain conditions, the student. If this data set is not found, it is assumed that any instructions in the LYRIC program requiring the use of the student data set will be treated as a "no operation."

Next, the LYRIC object deck is located and opened for use. Execution will start wither at the beginning or at a location indicated by status information in the student record.

The general construction of the executor is very similar to the assembler since there is a driving section and many utility and operand analysis subroutines.

Execution starts by branching to the statement extraction routine which reads in one object record as pointed to by the last branch address (set by TRUE or FALSE pointers depending on the execution of the last instruction). If this turns out to be an invalid record (not within the scope of the LYRIC data set) execution is terminated and control is passed back to the operating system. If the instruction defines the start of a new LYRIC step, an appropriate pointer is set. A default value for the next retrievable record is taken from the TRUE pointer in the object record. Control is passed back to the driving routine. The rest of the executor deals with the execution of each of the individual commands. If the command agrees with a list of legal commands, control is passed to that particular subroutine.

For example, GTO's would simply cause the record extraction routine to get a record from the location pointed to, in the instructions operand field. A test continuation card would load a buffer which, when filled (device dependent), would be dumped onto a display screen.

E. CONCLUSION

The remarks made above should suggest some of the major aspects of LYRIC in its present form.

A sample LYRIC program, a list of statements now implemented, and three key operands are presented in the sections which follow.

F. UCLA LYRIC SYNTAX AND DEFINITIONS

Capital letters must be used exactly as shown.

"text". Refers to any string of written information (including blanks) not exceeding 40 characters.

"number". Any integer not exceeding 255 (360 one byte constant).

"character string". Any series of 40 or less characters.

"(label)". An optional label not to exceed 5 characters in length.

"label". A required label not exceeding 5 characters in length.

"executable statement". This is, at present, considered any statement except additional text or GTO (see section on Operations and Their Formats.)

NOTE: All labels must be left justified in their fields. No more than 40 labels per program including system labels generated by each step using a PRO operand.

G. A SAMPLE LYRIC PROGRAM

PROBLEM: Ask a user if he would like to know how to use LYRIC. If 'yes' give him a positive response. If 'no' tell him he is 'negative' and he should type in 'yes'. If he does not type in either 'yes' or 'no' give him an error message and ask him the question again. If he is again wrong, terminate the program.

```
START PRE THIS IS A TEST OF LYRIC.
      PRO WOULD YOU LIKE TO LEARN MORE ABOUT
           LYRIC?  ANSWER YES OR NO.
      ANS
      KEY YES
          YOU INDEED REALIZE THAT LYRIC IS WELL
          WORTH LOOKING INTO.
      GTO NEX
      KEY NO
          YOU ARE DEFINITELY NEGATIVE.
          TRY A YES.
      GTO START
```

```
       UNX 1
           YOU DIDN'T TYPE YES OR NO.
           YOU HAVE ONE MORE CHANCE
       GTO PRO
       UNX 1
           YOU HAVE AGAIN FAILED TO ENTER A
           'YES' OR A 'NO'.
       GTO NEX
       PRE GOOD-BY
END    PRE
       END
```

H. OPERATIONS AND THEIR FORMATS

1. STATEMENT: PRE

ABSTRACT:

Defines the start of a screen of text and a LYRIC block.

SYNTAX:

label PRE text

DESCRIPTION:

A new LYRIC block is defined and the text in the statement field becomes the first text in the block and on the screen.

EXAMPLE:

```
PRE    This text does not require
       user intervention but you
       can ask for an ANS anyway.
ANS
```

COMMENTS:

(label) PRE ⟨text⟩

Present new step, no response required.

Used for display of information and can be the object of a NEX statement. Resets the 'current step' location.

2. STATEMENT: PRO

ABSTRACT:

LYRIC problem statement.

SYNTAX:

```
label PRO text
```

DESCRIPTION:

Defines the start of a LYRIC step requiring a response. It is exactly the same as a PRE since the ANS must always initiate an answer.

EXAMPLE:

```
sam      PRO    Hello there.
                I will help you.
                What did you say?
         ANS
         KEY    Ok.
```

COMMENTS:

(label) PRO ⟨text⟩
Problem statement with required response.
Similar to the PRE but implies the use of a trailing ANS.
Resets the 'current step' location.

3. STATEMENT: ⟨blank⟩

ABSTRACT:

Text continuation.

SYNTAX:

```
label ⟨ ⟩ text
```

DESCRIPTION:

Defines text as it will appear on a console screen. If branched to without any preceeding PRE or PRO's, the text is appended to the text on the screen with no break in flow.

EXAMPLE:

```
              PRE    The rain in
                     Spain
              GTO    SAM
               .
               .
               .
        SAM          falls mainly
                     on the plain.
```

COMMENTS:

(label)〈〉 〈text〉

Additional text.

May be used anywhere additional text must be produced.

4. STATEMENT: ANS

ABSTRACT:

Request an answer from a student beginning at a predefined location.

SYNTAX:

```
label ANS time_out_number,LOC=(column,row)
ARROW=(NO/column,row),CURSOR=(column,row)
```

DESCRIPTION:

The console will be placed in such a state that when the ENTER button is pressed the text typed in by the student will be read into an answer save area (ANS). If no options are specified an arrow will be placed after the previous PRE/PRO text with the cursor following it. The answer will be expected after this point. Answers can be read in from any point on the screen as defined by LOC=(column,row). The default arrow can be eliminated or repositioned, and the cursor can be positioned.

EXAMPLE:

```
PRO   The rain in spain falls where?
ANS   LOC=(1,3),ARROW=(1,3),CURSOR=(1,3)
PRO   what now?
ANS
ANS   10
      You have failed to enter anything.
GTO   MORE
```

COMMENTS:

This may require special programming for different consoles.

(label) ANS

Request an answer at this point.

A small arrow is displayed indicating the start of the user's answer. The console will then wait until information is typed in and the ENTER button is pressed. The screen is read, the answer field is loaded up to the cursor and execution continues. An answer may not be longer than 255 characters. Excess will be truncated.

5. STATEMENT: GUD

ABSTRACT:

Test the first characters of the input answer for an exact comparison against a correct answer.

SYNTAX:

```
label GUD text
```

DESCRIPTION:

If the answer is exactly the same as the operand up to the cursor, then give a positive result as in KEY.

EXAMPLE:

```
PRO   What color is the sky?
ANS
GUD   BLUE
      BLUE is correct
GTO   NEX
UNX   1
      The sky is BLUE.  You are incorrect.
```

COMMENTS:

(label) GUD ⟨text⟩
Test for exact correct answer.

Using the text field, up to but not including the last blank, the operand is compared with the user's reply. Only an exact comparison, starting directly after the arrow, will be considered correct. Operations proceed as in a KEY.

6. STATEMENT: KEY

ABSTRACT:

Search for a keyphrase in an answer.

SYNTAX:

```
label KEY 'keyphrase', 'keyphrase', 'keyphrase'...
```

DESCRIPTION:

Searches for a series of logically ORed keyphrases. As soon as one is found the search is terminated and the next sequential instruction is executed. and the required following GTO is executed. If no match is found, the instruction immediately following the next sequential GTO is executed. The search for each keyphrase starts at the first position of the answer field and ends at the N+L-1 position where N is the length of the answer field and L is the length of the keyphrase and the first position of the answer is at position 1. Each keyphrase is treated as a separate search (at the same level).

EXAMPLE:

```
same     KEY    'go','to','there'
                Excellent job.
         GTO    ON34
         GTO    NOGOOD
lab      KEY    'ain't','good' ????????
         GTO    GLITCH
         BAD    English
         GTO    BAD9
```

COMMENTS:

Each keyphrase in the statement will be searched for at the very start of the answer field. The translator could supply the location for each keyphrase in the statement as a constant. Lengths would also be helpful but a pre-TRT for the delimeter would do as well.

(label) KEY ⟨text⟩

Key word search.

Using the text field, up to but not including the last blank, searches the user answer for a match. If a match occurs execute the next sequential instruction. If not a match, go to the next non-continue, non-GTO statement. The scope of a KEY usually ends with a GTO display of information and can be the object of a NEX statement. Resets the current step location.

7. STATEMENT: BAD

ABSTRACT:

Test the answer for an exact comparison with an incorrect answer.

SYNTAX:

label BAD text

DESCRIPTION:

Exactly the same as a GUD.

EXAMPLE:

```
        BAD   HOW
              You said HOW.
        THIS IS NOT A QUESTION.
        GTO   NEX
        UNX   1
              You haven't done anything wrong yet
```

COMMENTS:

This statement will be exactly equivalent to the GUD statement. It exists only for the mnemonic value.

(label) BAD ⟨text⟩

Test for exact incorrect answer.

Acts exactly like GUD. Used for its mnemonic value.

8. STATEMENT: UNX

ABSTRACT:

Define the action in the event of an undefined answer.

SYNTAX:

```
label UNX number
```

DESCRIPTION:

The text of a UNX will be displayed each time the user passes through the UNX code while in the same LYRIC step. UNX's are executed when logically encountered in the same way as a KEY. They fail only when the statement has been exhausted. A UNX with a 0 statement is a no-operation.

EXAMPLE:

```
        KEY   YES
              Good.
        GTO   NEX
        UNX   3
              No no not that.
        GTO   PRO
        UNX   1
              Very bad.
        GTO   PRO
        KEY   Help.
        GTO   MORE
        PRE   Well here we are.
```

COMMENTS:

UNX's initialize a special save area whenever they are encountered by the executor. They may occur at any location in a LYRIC step. Leaving a step initializes the UNX save areas. This technique is compatible with almost any kind of implementation of the UNX function.

(label) UNX ⟨number⟩

Unexpected answer: In the event of an unexpected answer the following additional text records will be displayed. This display will occur each time the UNX is executed but not more than the number of times shown in the statement field. When the UNX is exhausted the next executable instruction is executed as in KEY.

9. STATEMENT: GTO

ABSTRACT:

Transfer control to the program location indicated.

SYNTAX:

label GTO/PHR/PRO/NEX/ANS/label/*

DESCRIPTION:

Causes control to be transferred to an explicitly or implicitly defined statement. The first GTO after the start of a LYRIC phrase terminates the phrase.

EXAMPLE:

```
                GTO  PHR
gto      MAN
                ANS
                GTO  ANS
                GTO  NEX
         joe    PRO  hi der
                GTO  PRO
```

COMMENTS:

A GTO should not force the dumping of the buffer or you can not concatenate text lines.

GTO <NEX/PRO/⟨label⟩>

Go to the indicated statement.

NEX = go to the next step: either the next sequential PRE or PRO encountered.

PRO = Branch to the first preceding PRO encountered.

⟨label⟩ = To any legal five character label.

10. STATEMENT: END

ABSTRACT:

Defines the end of a section of LYRIC code.

SYNTAX:

```
label END blank/'name of LYRIC processor module'
```

DESCRIPTION:

Defines the end of a section of LYRIC code. On the execution of an END, control is passed to the LYRIC program in the STATEMENT field, or, if the STATEMENT field is blank the student is signed out of the LYRIC processor.

EXAMPLE:

```
finish     END     'CSW015.SSS.LYR2'
end        END
```

COMMENTS:

If you branch to END, display the buffer first, then transfer control.
(label) END
End of program.

11. STATEMENT: PAC

ABSTRACT:

Remove specified characters from the answer field.

SYNTAX:

```
label PAC 'character string'
```

DESCRIPTION:

The individual characters in the statement are removed whenever they are found in the answer field.

EXAMPLE:

```
PRO     Type in 'encyclopedia'.
ANS
PAC     'aeiou'
KEY     NCCLPD
        very good.
GTO     MORE
UNX     1
        Not correct at all.
```

COMMENTS:

The data in the answer field is permanently changed by this instruction.

(label) PAC ⟨character string⟩

Compact the answer by eliminating the indicated characters. If the string starts with a blank or if the field is completely blank, the instruction will eliminate blanks. After the execution of this command, text manipulation commands must take into consideration the loss of compacted information.

12. STATEMENT: PHR statement

ABSTRACT:

A LYRIC phrase.

SYNTAX:

label opcode PHR

DESCRIPTION:

A LYRIC phrase is defined as the sphere of influence of a standard LYRIC text and display command (i.e. KEY). It is the location of the statement immediately following the next executable GTO ending the LYRIC phrase.

EXAMPLE:

```
                        KEY    sam
                               hello sam
                        GTO    phr
joe                     GTO    jim
The GTO is equivalent to saying:
                        GTO    jim
```

COMMENTS:

The main use of this will be to define LYRIC command extents.

13. STATEMENT: MOD

ABSTRACT:

Define device dependent considerations.

SYNTAX:

nolabel MOD device_information

DESCRIPTION:

Allows optimization for different display devices.

EXAMPLE:

```
MOD    2260,LINE=80
MOD    CCI,LINE=40
```

APPENDIX IV

ILR PROCESSING RECORD SPECIFICATION

by

Jay L. Cunningham
Institute of Library Research
University of California
Berkeley, California

ILR PROCESSING RECORD SPECIFICATION

By
Jay L. Cunningham,
Institute of Library Research
University of California
Berkeley, California

NOTICE: The following specification was prepared to document the preliminary version of an experimental bibliographic storage record (internal processing format) for computer use. It presents the record as of one point in time of an ongoing design. The reader should beware that extensive changes in data element definition and codes are taking place both at the Institute of Library Research where the record is being formulated, and at the Library of Congress where the MARC II Communications Format is being devised in preparation for a nationwide machine record distribution service. The ILR storage record will be compatible with the MARC II record format. For this reason, the present specification is to be considered a temporary working paper.

A. INTRODUCTION

This report provides the specification of the logical internal record design for the ILR File Organization Project. The record is designed to hold records for all types of library materials. At this writing the field definitions and codes comprise the elements related to monographs. In later phases, the design will be expanded to include elements pertinent to journal articles and serial titles. The record is not identical to the MARC II record. It is, however, intended to be convertible to and from MARC II. In that sense the ILR record appears quite similar in structure to the LC MARC II communications format.

This is a specification for programmer use and is not written as a detailed explanation of the record content. For details on the field definitions and codes, see the following two documents:

Avram, Henriette D., John F. Knapp, and Lucia J. Rather. The MARC II Format; a Communications Format for Bibliographic Data. Washington, D.C., U.S. Library of Congress, 1968. 6 p.

Cunningham, Jay L. Instruction Manual for Editorial Preparation of Catalog Source Data. Preliminary Edition. Berkeley, Institute of Library Research, University of California, 1968. 172 p.

B. PROGRAM NAME

This specification has been implemented in a computer program code-named "INFOCAL"--"INput for File Organization at U. of CAL." The program exists in Version 1 as of May 6, 1968. The program was written in PL/I, Version 3, Release 14 of F-level, OS/360.

C. NATURE OF THE RECORD

The data base in the File Organization Project is record oriented. That is, there will be one master record for each bibliographic entity. The initial entity to be recorded is a single book, or "monograph." The record design has two primary components: data elements and codes. The data are organized into groups of one or more data elements. Such a group is called a field.

The tag is the principal code used in the record. It both identifies the field by function and describes its type of content at the most general level. Example: Tag 100 identifies the field by function as MAIN ENTRY for a given record, and describes the field as containing the name of an author of the type PERSONAL NAME.

If a field contains more than one kind of data element, or if it contains two or more values representing a particular kind of element, the elements are separated by codes called sub-field delimiters.

A set of 7 auxiliary codes called indicators accompanies each field. At present, only two of these indicators are implemented. The indicators serve to provide additional information about the field.

An example is TAG 100, which will contain one of the following values in Indicator 1:

FIGURE 1: INDICATOR FOR MAIN ENTRY - PERSONAL NAME

Form of Name	Main Entry is not Subject	Main Entry is Subject
Forename	0	4
Single Surname	1	5
Multiple Surname	2	6
Name of Family	3	7

The relationships among data elements and codes used in the record are summarized in the following table:

FIGURE 2: STORAGE RECORD ORGANIZATION

Record Component	Content of Component	Type of Code Identifying the Component
field	data element (group of one or more)	tag indicators
sub-field	data element	delimiter

The ILR storage record contains all data input from a pre-coded catalog record, with the exception of supplementary data fields containing what are called "dashed-on entries," in cataloging terminology. Such groups of fields describe a supplement to a book, an index, appendices, or other related material which is usually bound separately and often is received after the main work for which the catalog record was created. The supplementary fields will probably be stored in a separate record linked to the main record by a number which is an extension of the master record number. A final decision has not been made regarding this problem at the time of writing.

D. RECORD COMPOSITION

The storage record is composed of a variable number of characters placed in contiguous byte locations.

A complete storage record is composed of four segments, as shown in Fig. 3. Each segment is composed of one or more fields. The fields are of either the fixed or variable-length type, depending on their function. The fields are organized into one or more data elements along with certain of their associated codes.* The bibliographic data elements are variable or fixed in presence. The codes are correspondingly fixed or variable in presence. The variable length data elements are in principle repeatable, although in practice not all are defined as repeatable. The codes are likewise repeatable in principle.

The four segments of the record are: the Leader (56 bytes); the Record Directory (variable length, 12 bytes per directory entry); the Fixed Length Data Elements (currently 26 bytes); and the Variable Fields (variable in occurrence, variable in length).

All byte positions in the fixed length segments of the record are expressed in terms relative to the first byte regarded as one. This was done only for the purpose of conforming to an early draft of the MARC II format, and will be changed in a future version of the conversion program (INFOCAL).

All fields which always appear in the storage record and which express the negative condition of a set of values, e.g., in the Fixed Length Data Elements, will be set to either zeros or blanks. The zero and blank carry meaning in these fields, usually as the default value. The zero will normally be set as the default in a binary-valued fixed length data element and blank will normally be set as the default in a multiple-choice fixed data elements. The latter includes elements which can have a range (e.g., one to three values).

---

*The tags and Indicators 1-5 are in the Directory. Indicators 6 and 7 are at the head of the variable field.

FIGURE 3:

SCHEMATIC OF ILR STORAGE RECORD, INFOCAL VERSION 1
(as of May 6, 1968)

Segment 1	Segment 2	Segment 3	Segment 4
LEADER	DIRECTORY	FIXED LENGTH DATA ELEMENTS	VARIABLE FIELDS
1 56	(N x 12)	1 26	

NOTE: As currently defined in the record design, Variable Field access is obtained by a scan of the Directory to find the Tag, then add contents of "Base Address of Data" to Starting Character Position of field desired. (S.C.P. is relative to the first character of the Fixed Length Data Elements.)

Access to the first character of the Fixed Length Data Elements can be obtained by finding the Base Address of Data at character positions 19-20 of the Leader, then use the value therein as a displacement from the beginning of the Leader.

The length of the Directory may be determined by a data element contained in character positions 48-50 in the Leader. It is called "Number of Entries in Directory." Its contents multiplied by 12 gives the length of the total Directory.

In the record currently implemented, there are no field terminators and no record terminator. Control is maintained by a total record length field in the Leader, and field lengths in the Directory Entry for each variable field (and for the Fixed Length Data Elements field).

E. LEADER

The Leader occupies the first 56 characters of every record. It contains elements describing and identifying the record, in contrast to the variable data fields, which describe a bibliographic entity (i.e., a monograph). Thus the leader tells the type of content included in the record, in terms of the form of library material represented; the components of the record structure in terms of the meanings of the tags and codes for the particular form of library material indicated; and the hierarchical level at which the record is pitched (e.g., for a monograph which is a member of a series, the record is for the monograph, not the series).

The program symbol name for the entire Leader is LIPREFX.

FIGURE 4:
ILR PROCESSING RECORD - SEGMENT 1, LEADER

PROGRAM SYMBOL	DATA ELEMENT	CHAR. POSITION	CONTENTS, REMARKS, ETC.
LILENGTH	Record Length	1-5	Total number of characters in record,stored as EBCDIC codes; right justified, with blank fill.
LISDATE	Status Date	6-11	Six-character date referring to LISTATUS. Currently all blanks.
LISTATUS	Record Status	12	One character. Contents: 0 - uncertified 1 - certified 2 - changed record 3 - deleted record
LILEGCNT	Legend Extension	13	Provides facility for extending record type (characters 14-17). Not currently used. Set to zero.
LITYPE	Record Type	14	One Character. Contents: a - book b - manuscript c - music (sheet) d - music (manuscript) e - maps and atlases f - maps (manuscript) g - motion pictures and filmstrips h - microfilm (original edition) i - phonorecords (spoken) j - phonorecords (music) k - pictures, etc. l - computer media m - other

(Continued on next page)

FIGURE 4 (Cont.):
ILR PROCESSING RECORD - SEGMENT 1, LEADER

PROGRAM SYMBOL	DATA ELEMENT	CHAR. POSITION	CONTENTS, REMARKS, ETC.
LIBLEVEL	Bibliographic Level	15-17	Contents: One to three characters, left justified, with blank fill. a - analytic c - collective m - monograph s - serial
LIINDCNT	Indicator count	18	Number of indicator bytes in a directory entry. Now set to 5.
LIBASE	Base Address of Variable Fields	19-20	The displacement to the first character of the fixed fields. A binary number equal to '56 + 12*n', where n is the number of entries in the directory.
LIORIGIN	Origin of Record	21-23	Three EBCDIC digits identifying the agency which keyboarded the record. '003' = ILR; '790' = U C Santa Cruz.
LIPDATE	Processor Date	24-29	Six character date referring to LIPROSOR. Currently all blanks.
LIPROSOR	Processor or Record	30-32	Three EBCDIC digits indicating the agency modifying/processing the machine record. If LIORIGIN is 790, LIPROSOR is 003.
LISOURCE	Source type of Catalog card	33	A code identifying the general source of the original catalog card. a - central (e.g., LC card or proofslip) b - local origin (original cataloging at LIAGENCY). c - NUC d - other library or source (e.g., Alanar)

(Continued on next page)

FIGURE 4 (Cont.):
ILR PROCESSING RECORD - SEGMENT 1, LEADER

PROGRAM SYMBOL	DATA ELEMENT	CHAR. POSITION	CONTENTS, REMARKS, ETC.
LIAGENCY	Agency of Source Type	34-36	Code for specific agency of LISOURCE, when it is known. Three EBCDIC digits. Intended for UC network use.
LIADAPTR	Adapter of Catalog Card	37-39	Three EBCDIC digits identifying the adapter of the catalog card when LISOURCE code is other than "b". If not known, set to blanks.
LINUMBER	Master Record Number (ILR-assigned)	40-46	EBCDIC master record number. Taken from cols. 1-6 of decklet. Character 40 is zero, currently.
LICHECK	Checksum on Record Number	47	A checksum on characters 40-46.
LIDIRLEN	Number Entries in Directory	48-50	EBCDIC digits with leading blanks.
LIDEOF	Date Entered on Master File	51-56	EBCDIC in the order 'mmddyy'. Now set to date of program execution.

[END OF LEADER]

F. DIRECTORY

The record directory is an index to the kind and location of the variable fields within the record. It contains a series of fields (called directory entries) which contain the tag numbers, the lengths of the variable fields, and the starting character positions of the fields. The directory entry is fixed in length, but the number of entries in a given record cannot be predetermined, so the directory as a whole is variable in length. The directory is automatically generated by the INFOCAL program.

The program symbol name for the entire directory is LIDRTRY.

FIGURE 5:
ILR PROCESSING RECORD - SEGMENT 2, RECORD DIRECTORY

	Length	Char. Pos.	Content
Directory Entry 1	3	0	**Tag (3 EBCDIC digits.)+**
	1	3	**Indicator 1* (EBCDIC character. If not used, blank.)**
	1	4	**Indicator 2* (Repeatable tag number, applicable to those tags which can appear more than once in a given record. If tag is not currently repeatable, indicator will be set to binary zero. An 8-bit binary digit.)**
	1 1 1	5	**Indicators 3,4,5 (Character positions provided for future expansion. Currently set to blanks.)**
	2	8	**Field Length (A 16-bit number giving the character length of the variable field, including Indicators 6 and 7.)**
	2	10	**Starting Character Position (A 16-bit number giving the position of the first character of the variable field. Currently the first character will always contain Indicator 6. This position is relative to the first character of the Fixed Length Data Elements Field (Segment 3).)**
	. . .		
Directory Entry n			
[End of Directory]			

The total length of the Directory Entry is 12 Characters. The total length of the Directory is 12 x the number of directory entries.

+See Fig. 7 for a list of tags and field names.

*For the most part, Indicator 1 has contents identical to that defined in the LC MARC II specification. Indicator 2 is a feature defined by ILR, but suggested by the Library of Congress for the purpose of insuring unambiguous access to data fields having duplicate tags in a record.

The Tags are sorted by the program on the first digit of the tag number only. The variable fields are stored in the order in which they were input (which generally corresponds to the order of appearance on the source catalog card).

The justification for storing some of the Indicators in the Directory and the rest in the Variable Field was as follows: Indicators which by their nature apply to the variable field as a whole (e.g., code for sub-type of name) were stored in the Directory. This renders them easily accessible when searching the record at the level of the Directory only, e.g., to test for the presence of certain conditions. A search for the variable field data itself thus will not have to be made in each case.

Indicators which by their nature apply only to parts of fields (or even one character), such as codes for diacritical marks, were placed within the variable field data itself. Such indicators will most likely be addessed only when there is actual need to process the variable field content.

G. FIXED LENGTH DATA ELEMENTS

Elements in the Fixed Length Data Elements Field are assigned fixed locations and lengths. The entire field may, in various types of records, assume a variable length, and therefore it has been given a tag number and a Directory Entry. For the initial version of the INFOCAL program, only monograph catalog records are being processed. The Fixed Length Data Elements for monographs have a field length of 26, currently.

The program symbol name for the entire Fixed Length Data Elements Field is LIFIXED. Its Tag No. is 000.

FIGURE 6:
ILR PROCESSING RECORD - SEGMENT 3. FIXED LENGTH DATA ELEMENTS

PROGRAM SYMBOL	DATA ELEMENT	CHAR. POSITION	CONTENTS, REMARKS, ETC.
LIDTYPE	Date Type	1	c - two dates, second is copyright m - two dates, second is terminal n - date not known q - digits missing in original date r - reprint s - single publication date

(Continued on next page)

FIGURE 6 (Cont.):
ILR PROCESSING RECORD - SEGMENT 3, FIXED LENGTH DATA ELEMENTS

PROGRAM SYMBOL	DATA ELEMENT	CHAR. POSITION	CONTENTS, REMARKS, ETC.
LIDATE1	**First or Only Date**	**2-5**	**Four digits, or blank if not known**
LIDATE2	**Second Date**	**6-9**	**Four digits, or blank if not present.**
LIMICROR	**Form of Reproduction**	**10**	**a - microfilm ƀ - none** **b - microfiche** **c - micro-opaque**
LIFORM	**Content Form**	**11-14**	**One to four, or no characters. Left-justified, with unused characters filled with blanks.** **a - bibliographies** **b - catalogs** **c - indexes** **d - abstracts** **e - dictionaries** **f - encyclopedias** **g - directories** **h - yearbooks** **i - statistics** **j - handbooks** **k - other** m - medical atlases
LIGOVPUB	**Government Publication Indicator**	**15**	**a - U.S. federal** **b - California state** **c - California county/municipal** **d - international** **e - other governments** **ƀ - none**

(Continued on next page)

FIGURE 6 (Cont.):
ILR PROCESSING RECORD - SEGMENT 3, FIXED LENGTH DATA ELEMENTS

PROGRAM SYMBOL	DATA ELEMENT	CHAR. POSITION	CONTENTS, REMARKS, ETC.
LICONPUB	Conference Publication Indicator	16	'0' = no '1' = yes
LIMEBODY	Main Entry In Body	17	'0' = no '1' = yes
LILITGRP	Literary Group	18	a - complete/collected works b - selected works c - prolific writer b̸ - none
LICNCELT	Cancel Title Added Entry in Dict. Cat.	19	'0' = no, don't cancel '1' = yes, cancel in dictionary catalog; make for div.
LICNTRY	Country of Publication	20-22	All blanks. Not currently implemented.
LIILLUS	Illustration Codes	23-26	One to four characters, left-justified with blank fill. Set by scan of Collation Input Field. a - illus. b - map c - portraits d - charts e - plans f - plates g - music i - coats of arms j - genealogical tables k - forms l - diagrams

[End of Fixed Length Data Elements]

H. VARIABLE FIELDS

The variable fields include all the standard bibliographic data elements defined for monograph catalog records, plus various control numbers pertaining to catalog records, such as the LC Card No. (The Fixed Length Data Elements Field, Tag 000, although not strictly a variable field, may vary in length from one kind of record to another. Because of its complexity, it was defined in detail above, in Part G.)

The variable fields contain data which by its nature is variable both in presence and in length. The fields are packed into the last portion of the storage record with no intervening gaps.

Currently, the individual variable field begins with two special-purpose character positions, Indicators 6 and 7, which are set to zero. These are provided for future use. It is intended to use them to control diacritical marks and similar special characters occurring in the field, and to control the applicability of the field to a given library on the basis of set-inclusion or exclusion codes.*

The principal identifying codes (tags, and indicators 1-5) for a given variable field are, as indicated previously, stored in the Directory Entry for the field, along with the field length and address of its starting character position.

A list of tagged fields and the data elements contained in each is presented in Fig. 7, below.

A list of the meanings of Indicator 1 which are currently implemented, is presented in Fig. 8.

This does not exhaust the coding supplied in the storage record. There is a third and last component, sub-field identification. This code, called a delimiter, serves to identify and describe particular data elements which may be contained in the tagged fields.

Delimiters are currently stored interspersed with the data in position ahead of the sub-field they identify. It is not certain whether this method will be retained in future versions of the storage format. Currently a "%" is stored as the delimiter symbol in storage records produced by INFOCAL. A chart listing the present contents of the delimited sub-fields in each tagged field is presented in Fig. 9.

---

*See Part J.

FIGURE 7:
VARIABLE FIELD TAGS AND DATA ELEMENTS
(as of May 6, 1968)

Tag	Variable Field Data Element
	CONTROL FIELDS
000	FIXED LENGTH DATA ELEMENTS FIELD
001	CONTROL NUMBER (LC CARD NO.)
002	LEGEND EXTENSION+
003	LANGUAGES
	CONTROL NUMBERS
010	LC CARD NUMBER*
011	NATIONAL BIBLIOGRAPHY NUMBER
012	STANDARD BOOK NUMBER**
013	PL 480 NUMBER
014	SEARCH CODE+
019	LOCAL SYSTEM NUMBER‡
	KNOWLEDGE NUMBERS
020	BNB CLASSIFICATION NUMBER**
030	DEWEY DECIMAL CLASSIFICATION NUMBER
050	LC CALL NUMBER
051	COPY STATEMENT (LC CARD)
052	CATALOGING SOURCE***
060	NLM CALL NUMBER
070	NAL CALL NUMBER**
071	NAL SUBJECT CATEGORY NUMBER
080	UDC NUMBER**
090	LOCAL CALL NUMBER (HOLDINGS)
091	COPY STATEMENT (LOCAL CARD)***
	MAIN ENTRY
100	PERSONAL NAME
108	TITLE SUBHEADING
110	CORPORATE NAME
111	CONFERENCE OR MEETING
112	FIRM NAME***
118	TITLE SUBHEADING
120	CORPORATE NAME WITH FORM SUBHEADING
128	TITLE SUBHEADING
130	UNIFORM TITLE HEADING
131	ANONYMOUS CLASSIC HEADING***
138	TITLE SUBHEADING
	SUPPLIED TITLES
200	UNIFORM TITLE
210	ROMANIZED TITLE
220	TRANSLATED TITLE**
	TITLE PARAGRAPH
240	TITLE STATEMENT
250	EDITION STATEMENT
	IMPRINT
260	PLACE
261	PUBLISHER
262	DATE(S)
300	COLLATION
350	BIBLIOGRAPHIC PRICE**
360	CONVERTED PRICE+
	SERIES NOTES
400	PERSONAL NAME (TRACED THE SAME)
408	TITLE SUBHEADING
410	CORPORATE NAME (TRACED THE SAME)

(Continued on next page)

FIGURE 7 (Cont.):
VARIABLE FIELD TAGS AND DATA ELEMENTS
(as of May 6, 1968)

Tag	Variable Field Data Element
411	CONFERENCE (TRACED THE SAME)
412	FIRM NAME***
418	TITLE SUBHEADING
440	TITLE (TRACED THE SAME)
490	SERIES UNTRACED OR TRACED DIFFERENTLY
	BIBLIOGRAPHIC NOTES
500	BIBLIOGRAPHY NOTE
510	DISSERTATION NOTE
520	CONTENTS NOTE (FORMATTED)
530	"BOUND WITH" NOTE
540	"LIMITED USE" NOTE
550	GENERAL NOTES (ALL OTHERS)
560	ABSTRACT**
570	"IN ANALYTIC" NOTE***
580	"FULL NAME" NOTES***
	SUBJECT ADDED ENTRY
600	PERSONAL NAME
608	TITLE SUBHEADING
610	CORPORATE NAME
611	CONFERENCE OR MEETING
612	FIRM NAME***
618	TITLE SUBHEADING
620	CORPORATE NAME WITH FORM SUBHEADING
628	TITLE SUBHEADING
630	UNIFORM TITLE HEADING
631	ANONYMOUS CLASSIC HEADING***
638	TITLE SUBHEADING
640	BOOK TITLE AS SUBJECT***
650	TOPICAL
651	GEOGRAPHIC NAMES
652	POLITICAL JURISDICTION ALONE OR WITH SUBJECT SUBDIVISIONS
653	PROPER NAMES NOT CAPABLE OF AUTHORSHIP
655	GENERAL SUBDIVISIONS (OTHER THAN PERIOD AND PLACE)
656	PERIOD SUBDIVISION
657	PLACE SUBDIVISION
660	NLM SUBJECT HEADINGS (MESH)
661	TOPICAL MESH SUBHEADINGS
662	GEOGRAPHIC MESH SUBHEADINGS
663	TIME PERIOD MESH SUBHEADINGS
664	FORM MESH SUBHEADINGS
670	NAL AGRICULTURAL/BIOLOGICAL VOCABULARY**
690	LOCAL SUBJECT HEADING SYSTEMS+
	OTHER ADDED ENTRIES
700	PERSONAL NAME
708	TITLE SUBHEADING
710	CORPORATE NAME
711	CONFERENCE OR MEETING
712	FIRM NAME***
718	TITLE SUBHEADING
720	CORPORATE NAME WITH FORM SUBHEADING
728	TITLE SUBHEADING
730	UNIFORM TITLE HEADING
731	ANONYMOUS CLASSIC HEADING***
738	TITLE SUBHEADING

(Continued on next page)

FIGURE 7 (Cont.):
VARIABLE FIELD TAGS AND DATA ELEMENTS
(as of May 6, 1968)

Tag	Variable Field Data Element	Tag	Variable Field Data Element
740	TITLE TRACED DIFFERENTLY		
753	PROPER NAMES NOT CAPABLE OF AUTHORSHIP		
	SERIES ADDED ENTRIES		
800	PERSONAL NAME		
808	TITLE SUBHEADING		
810	CORPORATE NAME		
811	CONFERENCE OR MEETING		
812	FIRM NAME***		
818	TITLE SUBHEADING		
840	TITLE-ONLY SERIES HEADING		
900	BLOCK OF 100 NUMBERS FOR LOCAL USE+		

NOTES:

* An alternate code defined in LC MARC II for optional local use. ILR will use Tag 001 for LC Card Nos. available to it, thus maintaining compatibility with the anticipated LC tape distribution service.

** Provided in LC MARC II for future use. ILR will supply this data in its records only if the information is already available on the LC cards used as input records at the time of original conversion.

*** A local code or data element specified for original conversions by ILR, but not defined in LC MARC II.

+ Code and data elements not currently implemented by ILR, but defined in LC MARC II.

‡ An alternate code defined in LC MARC II for optional local use. ILR will place its own locally-generated master record number in positions 40-46 of the storage record Leader.

FIGURE 8:
VALUES FOR INDICATOR 1 IN APPLICABLE FIELDS

Tag	Field	Indicator and Value		
003	LANGUAGES	Single or Multilanguage	0	
		Translation	1	
050	LC CALL NUMBER	Book in LC	0	
		Book Not in LC	1	
100	MAIN ENTRY (Personal Name)	Main Entry Not Subject		Main Entry is Subject
		Forename	0	4
		Single Surname	1	5
		Multiple Surname	2	6
		Name of Family	3	7
110 111	(Corporate Name) (Conference)	Surname (Inverted)	0	4
		Place + Name	1	5
		Name (direct Order)	2	6
112 120	(Firm Name) (Corporate Name with Form Subheading)		0	1
130	(Uniform Title Heading)		0	1
200	UNIFORM TITLE	Not Printed on LC Cards	0+	
		Printed on LC cards	1	
210	ROMANIZED TITLE	No Title Added Entry	0	
		Title Added Entry	1	
240	TITLE STATEMENT	No Title Added Entry	0	
		Title Added Entry	1	
261	PUBLISHER	Publisher Not Main Entry	0	
		Publisher is Main Entry	1	
400	SERIES NOTE (Personal Author/ Title)	Author Not Main Entry		Author is Main Entry
		Forename	0	4
		Single Surname	1	5
		Multiple Surname	2	6
		Name of Family	3	7

+This value not presently implemented.

(Continued on next page)

FIGURE 8 (Cont.):
VALUES FOR INDICATOR 1 IN APPLICABLE FIELDS

Tag	Field	Indicator and Value			
			Author Not Main Entry	Author is Main Entry	
410 411	(Corporate Author/ Title) (Conference/Title)	Surname (inverted)	0	4	
		Place + Name	1	5	
		Name (direct order)	2	6	
412	(Firm Name)		0	1	
490	(Series Untraced or Traced Differently)	Series Not Traced	0		
		Series Traced Differently	1		
600	SUBJECT ADDED ENTRIES (Personal)	Forename	0		
		Single Surname	1		
		Multiple Surname	2		
		Name of Family	3		
610 611	(Corporate) (Conference)	Surname (inverted)	0		
		Place + Name	1		
		Name (direct order)	2		
700	OTHER ADDED ENTRIES (Personal)		Alternative	Secondary	Analytical
		Forename	@	D	H
		Single Surname	A	E	I
		Multiple Surname	B	F	J
		Name of Family	C	G	K
710 711	(Corporate)	Surname (inverted)	@	D	H
		Place + Name	A	E	I
		Name (direct order)	B	F	J
712 720	(Firm Name) (Corporate with Form Subheading)		0	1	2
730 731	(Uniform Title Heading)		0	1	2
740	(Title Traced Differently)		0	1	2

FIGURE 8 (Cont.):
VALUES FOR INDICATOR 1 IN APPLICABLE FIELDS

800	SERIES ADDED ENTRIES (Personal Author/ Title)	Forename	0
		Single Surname	1
		Multiple Surname	2
		Name of Family	3
810 811	(Corporate Author/ Title)	Surname (inverted)	0
		Place + Name	1
		Name (direct order)	2

I. SUB-FIELD DELIMITER CODES

In the current version of the INFOCAL program, most sub-field delimiters have a form of coding as follows: each sub-field beyond the first is signified by a combination of one or more per cent (%) symbols in the storage record. That is, the sub-field coding is positional rather than explicit. Two %'s followed by a data string, for example, means the first and second sub-fields in the given field are vacant.

DATA PRESENT:	1st Sub-field data%2d Sub-field data%3d Sub-field data
DATA ABSENT:	%%3d Sub-field data

Because this area is the most volatile one in terms of changes in the form of the coding in the MARC II record design, both the presently implemented INFOCAL coding and the latest known MARC II coding are presented, for comparative purposes. This will assist retrofit of INFOCAL in the future.

It should be noted that changes in the coding imply no change in the kind or amount of bibliographic data included in the record. Full data will be included in all cases. The changes relate to the method of coding and the amount of detail in data element definition. Since the changes in sub-field definition affect field tags, the programmer should consult the latest MARC II record specifications issued by the Library of Congress before proceeding with any changes to the coding produced by INFOCAL. (Certain of the data elements currently defined in INFOCAL as tagged fields are redefined in the new MARC II record as delimited sub-fields. Also, certain data elements have been defined as sub-fields by MARC II which have not yet been implemented in INFOCAL).

The revised system of sub-field coding announced by the Library of Congress, as currently understood, is as follows:

> In a field in which only one data element has been defined (e.g., Standard Book Number), the field will begin with the delimiter code "$a". The same data element may be repeated in a field as many times as necessary by preceding it with its identifying delimiter. For example, there may arise a need to assign more than one Standard Book Number to a single catalog record.
>
> Only fields which can contain more than one kind of defined data element are listed below.

FIGURE 9:
SUB-FIELD DELIMITER CODES

NOTE: The word "None" in this table means that the data element was not defined by ILR for sub-field coding purposes at the time the INFOCAL program was implemented.

Sub-field Code in INFOCAL Storage Record	Revised MARC II Code	TAG, FIELD NAME, AND SUB-FIELDS	
		003	Languages
None	$a	The group of 3-character language codes needed to describe the languages of the text or its translation	
%	$b	Summaries	
		Example: $aengfre$bgerrus	
		050 060 070	LC, NLM, and NAL Call Numbers
None	$a	Class Number	
None	$b	Book Number	
		051	LC Copy Statement
		091	Copy Statement (LOCAL CARD)
None	$a	Class Number	
None	$b	Book Number	
%	$c	Copy Information	
	(Not defined in MARC II)	052	Cataloging Source
None		Name of Library Contributing Cat. Copy	
%		Class No./Call No. (when present)	
		090	Local Library Holdings
None	$a	Call Number	
%	(Not def. in MARC)	Copy Number	
%	$b	Library Code	
%	$c	No. of Copies	

(Continued on next page)

FIGURE 9 (Cont.):
SUB-FIELD DELIMITER CODES

Sub-Field Code in INFOCAL Storage Record	Revised MARC II Code	TAG, FIELD NAME, AND SUB-FIELDS
		100 400 600 700 800 Personal Name
None	$a	Name
None	$b	Numeration
%	$c	Titles and other words
%	$d	Dates
%	$e	Relator
None	$k	Form Sub-heading
Tag*	$t	Title (of book or title series)
None	$u	Filing Information (in 700 only)
%	$v	Volume or No. (in 400/800 only)
		110 410 610 710 810 Corporate Name (also for 112/412/612/712/812 which are ILR tags for firm names) (also for 120/620/720, Corp. Name with Form Subheading)
None	$a	Name
%	$b	Each subordinate unit in hierarchy
#	$k	Form Sub-heading
Tag	$t	Title (of book or title of series)
None	$u	Filing Information (in 710 only)
%	$v	Vol. or No. (in 410/810 only)
		111 411 611 711 811 Conference or Meeting
None	$a	Name

*The word "Tag" here means that the data element is identified in the current version of INFOCAL as a tagged field. See Fig. 7.

FIGURE 9 (Cont.):
SUB-FIELD DELIMITER CODES

Code in INFOCAL Storage Record	Revised MARC II Code	TAG, FIELD NAME, AND SUB-FIELDS
%	$b	Number
%	$c	Place
%	$d	Date
None	$e	Subordinate unit in Name
None	$g	Other information in the heading
#	$k	Form Sub-heading
Tag*	$t	Title(of book or title of series)
None	$u	Filing Information (in 711 only)
%	$v	Vol. or No. (in 411/811 only)
		130 630 730 — Uniform Title Heading (also for 131/631/731, Anon. Classic heading - an ILR tag)
None	$a	Uniform title (Main portion)
Tag	$t	Title (of part)
None	$u	Filing Information (730 only)
		245 — Title Page Title
None	$a	Short title from which added entry is made
%	$b	Remainder of title
%	$c	Remainder of title page transcription
		250 — Edition Statement
None	$a	Edition
%	$b	Additional information after edition
		260 — Imprint
Tag	$a	Place
Tag	$b	Publisher
Tag	$c	Date
		300 — Collation Statement
None	$a	Pagination
%	$b	Illustrative Matter
%	$c	Size
None	$d	Thickness

*The word "Tag" here means that the data element is identified in the current version of INFOCAL as a tagged field. See Fig. 7.

FIGURE 9 (Cont.):
SUB-FIELD DELIMITER CODES

Sub-Field Code in INFOCAL Storage Record	Revised MARC II Code	TAG, FIELD NAME, AND SUB-FIELDS	
		440	Series Note Traced Same (Title)
None	$a	Title Portion	
%	$v	Vol. or No.	
		650 651 652 653 654	Subject Added Entries
None	$a	Main Subject Heading (650-654 only)	
Tag*	$x	General Subject Subdivisions	(also 600/610/611/612 and 630)
Tag	$y	Period Subject Subdivisions	
Tag	$z	Place Subject Subdivisions	
Tag	$t	Title portion of a subject heading	
		840	Series Traced Differently (Title)
None	$a	Title of series	
%	$v	Vol. or No.	

*The word "Tag" here means that the data element is identified in the current version of INFOCAL as a tagged field. See Fig. 7.

J. ADDITIONAL VARIABLE FIELD FEATURES

Two character positions (Indicators 6 and 7) have been placed at the head of each variable field occurring in the record, beginning with Tag 002, Legend Extension.

These indicators serve as the nucleus for expansion of the leading portion of the variable field to allow encoding of three types of information which can be applied to the field when it is prepared for output display: the alphabet of the field, location-specific information, and a list of diacritical marks. A proposed encoding sequence is shown in Fig. 10. (These features have not been implemented in INFOCAL).

FIGURE 10:
PROPOSED VARIABLE FIELD HEADER

START OF FIELD	IND. 6	IND. 7	DATA
alphabet code	location-specific codes	diacritics list	variable field content

1. Alphabet Code. A flag to shift the encoding of the field from the English alphabet to some other character set such as Cyrillic. It would specify both the fact of the shift and the particular encoded alphabet. An unresolved problem is that a deshift back to English or to a third alphabet can occur within the field. The flag must be kept distinct from the first byte of the location-specific information. The alphabet code could be omitted in the default case of "all-English."

2. Location-Specific Codes. Indicator 6 has been reserved as a flag to show whether a given field content is desired by a given library to which the record applies. For example, the record might contain both LC subject headings and Medical Subject Headings. Library A might want only the LC headings; library B only the MeSH headings. This Indicator will facilitate the composition and display of the record according to each library's specifications. Four configurations are possible:

a. If the field is wanted by all locations, the indicator is to be set to $F0_{16}$ (currently, perhaps it should be 00). This would be the default case. The next character in sequence is the diacritics indicator (Ind. 7).

b. If all but certain locations are to be provided with the field, the indicator will be set to a stacked code of 40_{16} plus the number of location codes which follow. Location codes are set as three-digit EBCDIC numbers, following Indicator 6 and preceding Indicator 7.

c. If only certain locations are to be provided with the field, the indicator will be set to a stacked code of 80_{16} plus the number of location codes which follow.

d. If none of the libraries are to be supplied with the field, a code of 00_{16} is set. This will be rarely used, and will perhaps apply to information used only by the processor of the records--e.g., the ILR master record number, if stored in a variable field, would not conceivably by desired as a "print" field by any libraries to which the record applies.

The location-specific code will always be present.

3. Overprinted Character Coding. Indicator 7 is the nucleus for a proposed method of storing diacritical marks and any other characters which are overprinted and thus require special processing, or which might well be omitted from certain kinds of displays, e.g., CRT's. The major objective of the code will be to preclude a character-by-character scan of the field content to determine if overprinted characters exist. Rather a simple test on the indicator will yield the desired information. The text string in the field will thus not be interrupted by special codes, and no special processing will be necessary should it be desired to bring the field out for display without the overprinted characters.

The proposed technique is as follows:

a. Each variable field will have a one-byte signal in Indicator 7, which will always be present. It will be followed by 0-n sub-fields, each containing a value corresponding to a diacritical present in the field, and a pointer (a relative character position).

b. If the signal is set to zero, there are no special characters, and the variable field data is free of special handling. Data starts in Field + 2.

IND. 6	IND. 7	
	0	Data

c. If the signal is set to an even-numbered multiple n (n=2→ 254) there are n/2 diacriticals in the field's data. The signal will be set to the relative character position in the field of the first data character. The signal will be followed by one or more sub-fields which will show the value of the diacritic together with a pointer to the data character to which the diacritic is to be applied. (The value comes first so that the program can be set up to determine whether or not an actual operation on the data should be subsequently performed.) The second pointer is counted from the position of the text character to which the preceding diacritic was applied. Etc.

	n	Value 1	Pointer 1	Value 2	Pointer 2	Data ...

E.g.,

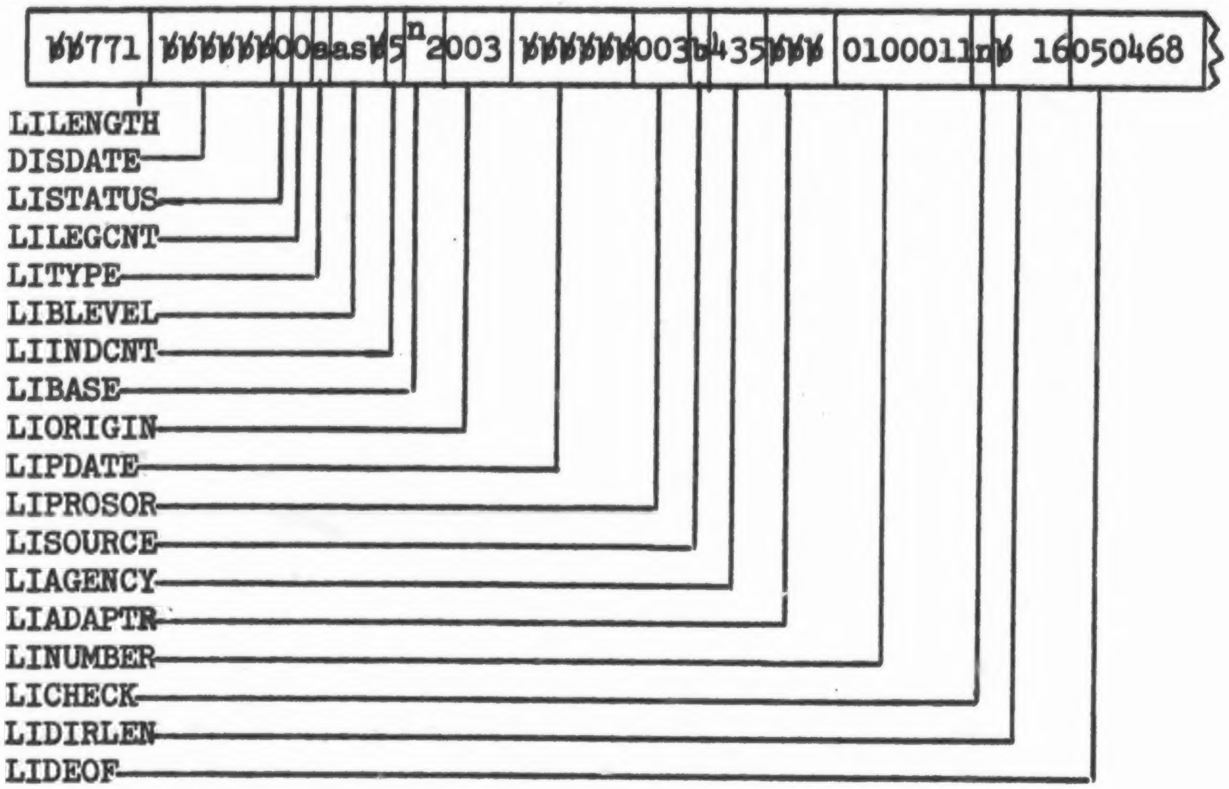
4	´	3	`	2	Therese ...

d. If the field length exceeds 255, and one or more diacritics occurs at a position beyond the 255th relative position, the signal will be set as above, but the first (or subsequent) sub-field will be set to blank as a Value, and 255 as a Pointer to serve as a displacement value. The next sequential subfield will show the relative position beyond 255 of the character to which the next diacritic applies.

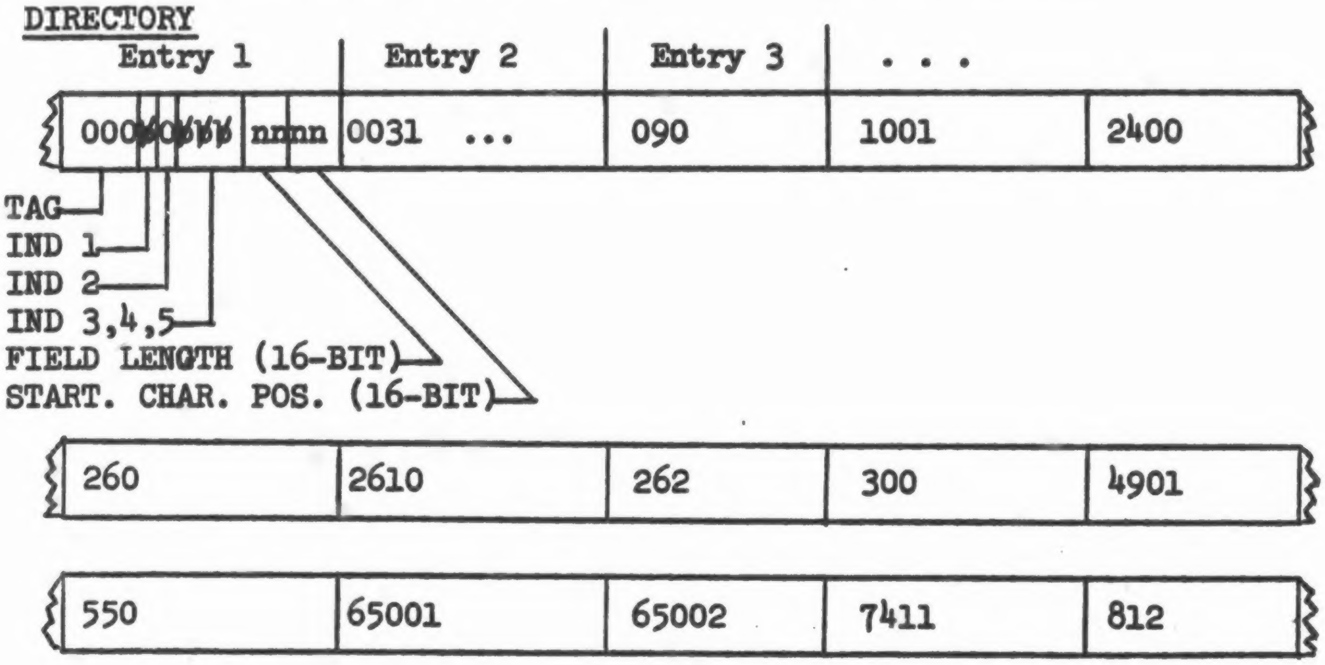
	4	Value 1 b̸	255	Value 2	5	Data ...

K. EXAMPLE OF STORAGE RECORD

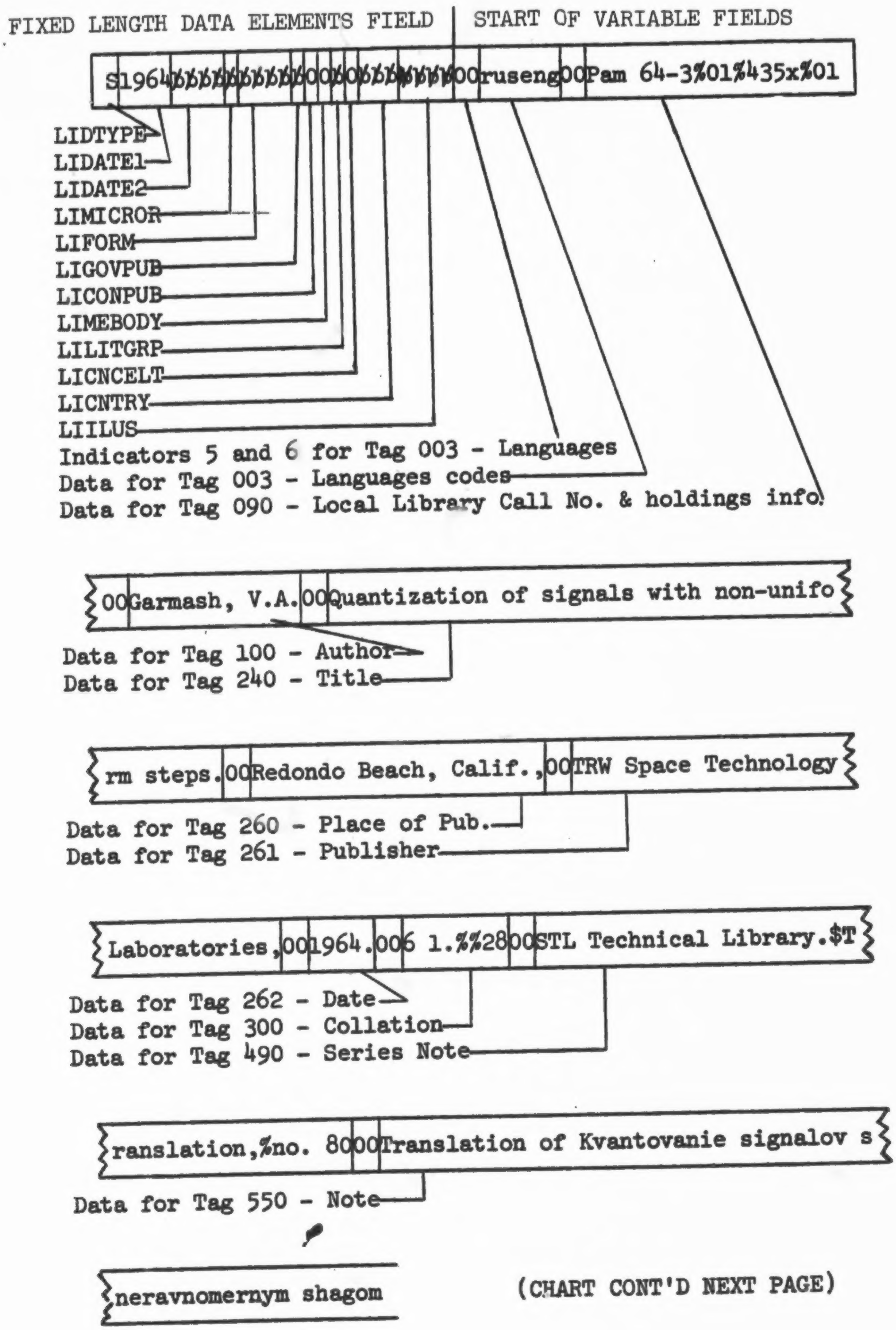
LEADER


b̸b̸771
b̸b̸b̸b̸b̸b̸00aasb̸5
n
2003
b̸b̸b̸b̸b̸b̸003b435b̸b̸b̸
010001lnb̸
16050468
LILENGTH
DISDATE
LISTATUS
LILEGCNT
LITYPE
LIBLEVEL
LIINDCNT
LIBASE
LIORIGIN
LIPDATE
LIPROSOR
LISOURCE
LIAGENCY
LIADAPTR
LINUMBER
LICHECK
LIDIRLEN
LIDEOF

DIRECTORY


Entry 1
Entry 2
Entry 3
. . .
000b̸0b̸b̸b̸ nnnn
0031 ...
090
1001
2400
TAG
IND 1
IND 2
IND 3,4,5
FIELD LENGTH (16-BIT)
START. CHAR. POS. (16-BIT)
260
2610
262
300
4901
550
65001
65002
7411
812

END OF DIRECTORY (CHART CONT'D NEXT PAGE)


FIXED LENGTH DATA ELEMENTS FIELD
START OF VARIABLE FIELDS
S1964ƀƀƀƀƀƀƀƀƀƀ00ƀ0ƀƀƀƀƀƀƀ00ruseng00Pam 64-3%01%435x%01
LIDTYPE
LIDATE1
LIDATE2
LIMICROR
LIFORM
LIGOVPUB
LICONPUB
LIMEBODY
LILITGRP
LICNCELT
LICNTRY
LIILUS
Indicators 5 and 6 for Tag 003 - Languages
Data for Tag 003 - Languages codes
Data for Tag 090 - Local Library Call No. & holdings info.
00Garmash, V.A.00Quantization of signals with non-unifo
Data for Tag 100 - Author
Data for Tag 240 - Title
rm steps.00Redondo Beach, Calif.,00TRW Space Technology
Data for Tag 260 - Place of Pub.
Data for Tag 261 - Publisher
Laboratories,001964.006 1.%%2800STL Technical Library.$T
Data for Tag 262 - Date
Data for Tag 300 - Collation
Data for Tag 490 - Series Note
ranslation,%no. 8000Translation of Kvantovanie signalov s
Data for Tag 550 - Note
neravnomernym shagom

(CHART CONT'D NEXT PAGE)

from Elektrosvyaz, no. 10, p. 10-12, 1957.|00|Information

Data for Tag 650 - Subject Heading (1st)

measurement.|00|Signals and signaling.|00|Elektrosvyaz,%v. 10,

Data for Tag 650 - Subject Heading (2d)
Data for Tag 74 - Added Entry for Periodical

p. 10-12, 1957.|00|Space Technology Laboratories, Inc, Lo

Data for Tag 812 - Series-Traced-Differently Added Entry

s Angeles.%Technical Library.|00|Translation,%no. 80

Data for Tag 818 - Series title

END OF RECORD

The catalog card shown below was used as the source for the preceding example of a storage record.

Eng.-Math.
Sciences
Pam
64-3

Garmash, V A

Quantization of signals with non-uniform steps. Redondo Beach, Calif., TRW Space Technology Laboratories, 1964.

6 ℓ. 28 cm. (STL Technical Library. Translation, no. 80)

Translation of Kvantovanie signalov s neravnomernym shagom from Elektrosvyaz, no. 10, p. 10-12, 1957.

1. Information measurement. 2. Signals and signalling. I. Elektrosvyaz, v.10, p. 10-12, 1957. II. Title. (Series: Space Technology Laboratories, Inc., Los Angeles. Technical Library. Translation, no. 80)

UCLA

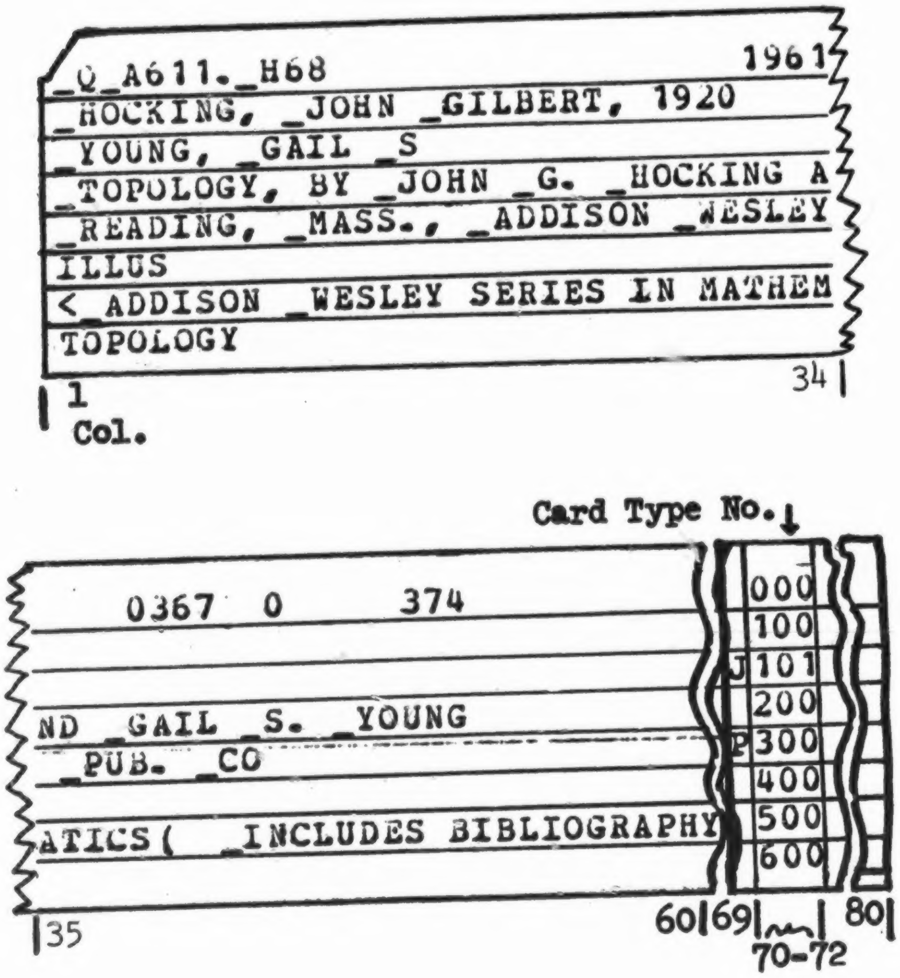
APPENDIX V

SUMMARY OF RECORD FORMATS FOR DATA BASES TO BE CONVERTED TO ILR PROCESSING RECORD FORMAT

1. Santa Cruz Record Format
2. ILR Input Record Format
3. Experimental On-line Mathematics Citation Data Base

V - 1. SANTA CRUZ RECORD FORMAT

FIGURE 1:
SAMPLE CATALOG RECORD IN ORIGINAL SANTA CRUZ FORMAT


_Q_A611._H68 1961
_HOCKING, _JOHN _GILBERT, 1920
_YOUNG, _GAIL _S
_TOPOLOGY, BY _JOHN _G. _HOCKING A
_READING, _MASS., _ADDISON _WESLEY
ILLUS
< _ADDISON _WESLEY SERIES IN MATHEM
TOPOLOGY
1
Col.
34
Card Type No.
0367 0 374
ND _GAIL _S. _YOUNG
_PUB. _CO
ATICS(_INCLUDES BIBLIOGRAPHY
J
P
000
100
101
200
300
400
500
600
35
60
69
70-72
80

The above is an example of a catalog record punched in the local input format devised at the University of Santa Cruz Library in 1965. From this kind of card-image file, the conversion program being written by ILR will produce card images reformatted into the ILR standard input format for subsequent processing by the program entitled "INFOCAL." The INFOCAL output will be the same record in ILR Storage Format (see Appendix IV). The latter record is the one that is convertible to and from the MARC II communications format devised by the Library of Congress.

The following pages constitute a summary of the card formats for each card type in the Santa Cruz record. The information has been excerpted from: Computer Usage Co., Inc. Specification for an Automated Library System. Prepared for University of California at Santa Cruz. Palo Alto, Calif., April, 1965. pp. 51-60.

A. SHELF KEY CARD - CARD TYPE NO. 000

FIELD ID	FROM COL	TO COL	COMMENTS
Call number	1	24	
Series	25	-	Blank, Alpha, or Numeric
Volume	26	28	Blank or 1 - 999, Left justify
Part number	29	30	Blank or 1 - 99, Left justify
Year	31	34	Year of publication
Donor number	35	38	Blank or Donor number from Gift list
Date rec'd.	39	42	Month and year received
Location	43	44	Alpha code designating Campus or Library
Type Code	45	-	0 = book 2 = reference 4 = see author 1 = serial 3 = govt. pub. 5 = see subj. 6 = see also subj. o'pch'-' = So. Pacific
Language Codes	46	50	Up to 5 Alpha Language codes, or 4 codes followed by '+'
Pages	51	55	Left justify
Correct/ Delete	56	-	Delete old records if 'C' or 'D' Substitute new records if 'C'
	57	69	- unused -
Card No.	70	72	'000'
	73	75	Currently unused, but available as part of Accession number field
	76	80	Accession Number

4

B. PERSONAL AUTHOR - CARD TYPE NO. 100-104

Limit: 0 - 5 Cards

FIELD ID	FROM COL	TO COL	COMMENTS
Author	1	60	name of author - left justified
	61	68	- unused -
Special Code	69	-	- see page following corporate author card -
Card No.	70	72	100 thru 104
Accession No.	73	80	same as Shelf Key card

Note: The 1st Author to be processed by the computer is considered the main author. The Main Author appears on all catalogs and is represented in the title by "**".

C. CORPORATE AUTHOR - CARD TYPE NO. 110-119

Limit: 1 or 2 cards per author/ 0 - 5 Authors

FIELD ID	FROM COL	TO COL	COMMENTS
Corporate Author	1	60	May be continued on second card
	61	67	- unused -
Continuation Indicator	68	-	is '-' if author is continued on second card
Special Code	69	-	- see next page -
Card No.	70	72	110 - 119
Accession No.	73	80	same as Shelf Key Card

See note concerning main author on Personal Author card.

SPECIAL CODES FOR AUTHORS

Type 1: To create added notation on Author Catalog

CODE	MEANING	NOTATION
J	Joint Author	joint auth.
C	Compiler	comp.
E	Editor	ed.
G	Joint Editor	joint ed.
I	Illustrator	illus.
P	Publisher	publ.
T	Translator	trans.

Type 2: To specify a substitute sort key.

X	Use this author as a substitute sort key for previous author. Previous author will appear on appropriate catalog but this author will not.

D. TITLE CARD - CARD TYPE NO. 200-224

Limits: 1 to 5 cards per title/ 0 - 5 Titles

FIELD ID	FROM COL	TO COL	COMMENTS
Title	1	60	may be continued on up to 4 additional cards
	61	67	- unused -
Continuation Indicator	68	-	is '-' if continued on next card
Special Code	69	-	- see next page -
Card No.	70	72	200-224
Accession No.	73	80	same as Shelf Key card

SPECIAL CODES FOR TITLES

CODE	MEANING
(blank)	Suppress listing this title in TITLE catalog
T	Title is a transliterated title
S	Title is a series title
P	Partial title
D	Standard title
L	This title is to be listed in TITLE catalog

Note: In all cases, the 1st title encountered when processing a given entry will be the only title which appears in the Shelf, Author, and Subject Catalog.

E. PUBLISHER/SOURCE CARD - CARD TYPE NO. 300-305

Limit: 1 - 2 cards per publisher/source/3 publ./sources total

FIELD ID	FROM COL	TO COL	COMMENTS
Publisher Source	1	60	may be continued on a second card
	61	67	- unused -
Continuation Indicator	68	-	is '-' if publ./source is continued on next card
Special Code	69	-	P = publisher S = source
Card No.	70	72	300-305
Accession No.	73	80	same as Shelf Key card

F. COLLATION CARD - CARD TYPE NO. 400

FIELD ID	FROM COL	TO COL	COMMENTS
Collation	1	60	as desired
	61	69	- unused -
Card No.	70	72	400 (1 card only)
Accession No.	73	80	same as Shelf Key card

G. COMMENTARY CARD - CARD TYPE NO. 500-509

Limit: 1 - 5 cards per comment/2 commentaries (1 of each type)

FIELD ID	FROM COL	TO COL	COMMENTS
Commentary	1	60	may be continued on up to 4 more cards
	61	67	- unused -
Continuation Indicator	68	-	is '-' if commentary continued on next card
Special Code	69	-	'S' for commentary to appear on shelf list only
Card No.	70	72	500-509
Accession No.	73	80	same as Shelf Key card

Note: If 2 commentary entries are used, at least one must have an 'S' code.

H. SUBJECT CARD - CARD TYPE NO. 600-604

Limit: 1 card per subject/5 subjects

FIELD ID	FROM COL	TO COL	COMMENTS
Subject	1	60	as desired
	61	68	- unused -
Special Code	69	-	may be used to indicate level of subject*
Card No.	70	72	600-604
Accession No.	73	80	same as Shelf Key card

*the special code is ignored by the system at present time.

V-2. ILR INPUT RECORD FORMAT

By

Jay L. Cunningham
Institute of Library Research
University of California
Berkeley, California

A. INTRODUCTION

In this report, a specification for a bibliographic data input record is presented. It was developed for use in converting catalog cards for monographs on an original basis - i.e., when no machine record is found to exist or expected to be obtainable elsewhere.* Because the format is experimental, the specification is presented in two parts: 1) the preliminary version, which has been programmed and is operational in a prototype production for test and evaluation purposes, and 2) suggestions for a revised and improved version which has not been programmed but which is based on the experience gained from the initial model.

The principal conclusions drawn from the evaluation of the prototype input format for this project are that a record of the complexity and variability of the MARC II record (to which the input record is convertible) is very costly to prepare. It is difficult to recruit and retain personnel of the quality appropriate to this work yet suited to its semi-clerical nature. Preliminary estimates are that average editing time per record will range from about 2-1/2 minutes to 4 minutes per man-record. The coding required by the MARC II format is the principal reason for these figures. That is quite aside from the unavailability at this time of an editing manual from the Library of Congress, and the fact that the MARC II design is itself still developmental (with the result that the instructions for editors are continually subject to revision). The reasons lie elsewhere: in the ill-structured nature of cataloging itself, in the ambiguities of the rules and the consequent ambiguities built into MARC (which is based on the rules), and the variance over the years of these rules and the reflection of this in catalog cards. Retrospective cards in particular have caused considerable delay in the project. Adjustments in the coding prescribed by MARC (which was aimed at current and future records) have repeatedly been necessary when older records are to be converted.

Nevertheless, these problems comprise a small percentage of the records. Therefore, the source of improvement does not seem to lie in a reduction of the MARC design to a less complex level. Rather the approach has been to utilize the MARC data definition to the fullest degree, in the expectation that the residue of coding interpretation problems will disappear as the format is "debugged." Only the form of coding has been altered in our input format.

---

*The writer is indebted for the great deal of groundwork on this record design done by Kelley Cartwright, formerly of the Institute staff at Berkeley, and Miss Martha Bovee, of the Technical Services Staff, University Library, University of California, San Diego.

We are employing a more concise notation in the coding, exploiting the concept of "default" settings, and building computer algorithms for the recognition and identification of the data elements in catalog records. The improved version of the input format extends this approach, and reflects a number of recommendations that we now feel confident can be implemented by program, as a replacement for a certain amount of manual editing effort. These program routines would operate either by simple key word matching or by more complex data recognition algorithms which process clues imbedded in the fields. The new format takes advantage of a minimum of human-applied codes that serve as boundary markers for the operation of the algorithms. Where possible these markers would be set by actions such as paragraph indention symbols keyed by the device operator without detriment to a normal typing rhythm.

B. NATURE OF THE MACHINE RECORD

1. General. As implied above, the input record format is compatible with the MARC II Communications format. Strictly speaking, the input record is converted in our system to the ILR Processing format. The latter is convertible to and from MARC II. For all of these record formats, however, the machine record organization can be summarized as follows: the two primary building blocks are data elements and codes. The data elements and codes are organized into the components shown in Fig. 1. The field is the basic component for processing purposes.

FIGURE 1:

STORAGE RECORD COMPONENTS AND ORGANIZATION

Record Component	Content of Component	Type of Code Identifying the Component
field	data element (group of one or more)	tag+indicators
sub-field	data element	delimiter

It is important to understand the reasons for the organization of the record into these components, in order to see why the input format while identical in content, is different in structure and coding from the other formats.

The MARC record coding is intended to serve four basic functions:

1. printing
2. sorting
3. information retrieval
4. catalog division

In the simplest design to serve these functions each data element in the computer record could have been defined as occupying its own field. Each field would have an identifying code or "tag." Due to the desire to use a record directory technique in the Processing and Communications Formats, the use of a field tag for each value of an element occurring in a given record becomes inefficient, since each tag is stored in a directory entry of 12 characters in length. (This constraint does not apply to the input record, since it does not have a directory of its own contents.)

Moreover, the data elements in many cases naturally group in clusters that will respond to the functions mentioned above. Many elements are both derived and manipulated in combination; for instance, the Imprint field comprises place, publisher and date of publication. It is thus logical to group more than one element in a field. There is still the requirement to identify individual elements, however, so a form of sub-field identification, less clumsy than the tag and its directory information was created. These are called delimiters and are embedded in the variable data. (The tag is relegated to a separate directory section of the logical record).

The ILR input format takes no intentional cognizance of the four functions. Its only requirement is to uniquely but concisely identify each element so that it may be transformed into another code which itself is more efficient for either internal processing or communications. It acts only as a one-directional data conversion vehicle from source to machine edit program. As a consequence the codes used in the input format must be adjusted to the structural patterns - options, alternations, repetitions - in catalog data in different ways than the processing format codes.

These field patterns arise from the ways the values for bibliographic data elements are organized in a given catalog record:

- Single occurrence of a value for a single element field, e.g., LC Card Number;
- Repeated values for a single element, e.g., several Dewey numbers on one card;

- Sequences of values for two or more different data elements, e.g., place, publisher, date;
- Variability in presence of values in these sequences, e.g., place, date;
- Combinations of the above, e.g., place, place, publisher, date, date; place, publisher, date, place, publisher, date.

In the input format, convenience in editing demands that complex patterns of elements in dense catalog text be coded as concisely as possible. Input tags for a pattern such as "place, place, publisher, date, date" may be coded as

/place,$place,/publisher,/date,date

The slashes identify the data by ordinal position, i.e., the 5th slash in an input string is always 'place'. In the processing format (MARC II) each of the above symbols might be coded by a three digit tag and generate a directory entry. The MARC design, which is still evolving, tries to balance the assignment of the tag level of format definition to a data element by consideration of such factors as:

- Is the data element processed frequently?
- If so, is it searched for or manipulated independently or in combination with other elements?
- What is the average length of the values of the data element?
- How often do sequences of repeated values of the element occur?
- Does it occur in every record?
- What is its format function (does it convey information about the document, about the record, about the file or even about other fields or codes in the record)?

Fig. 2 shows the structural patterns in the revised coding for MARC records. From this, it is evident that our input record represents a more primitive coding. This is acceptable in this format because the heavy volume of input symbol-scanning and syntactical checking is done only once: when the input compilation is performed. Once the internal record with its apparatus of field starting position pointers, field length counters, end-of-field/record/file signals etc. has been constructed, the user may process the data in strings and with table-driven routines.

FIGURE 2:

STRUCTURAL PATTERNS IN MARC RECORD DATA DEFINITION

		Single-Element Field		Multiple-Element Field	
		Single Value	Mult. Value	Single Value	Mult. Value
Repeatable Field Code	Repeatable Sub-Field Code	—	—	—	1*
	Non-Repeatable Sub-Field Code	9	7	5	2
Non-Repeatable Field Code	Repeatable Sub-Field Code	—	8	—	3
	Non-Repeatable Sub-Field Code	10	—	6	4

*Entries correspond to explanations and examples immediately following the diagram.

STRUCTURAL PATTERNS

Multiple Element Field, Multiple Values.

1. *Field Repeatable, Sub-field Repeatable.*

   Example: 6XX--Subject Tracings, with Subject Subdivisions.

   Form: $aMain Subject Heading$xSubject Subdivision-$xSubject Subdivision$xSubject Subdivision.

2. *Field Repeatable, Sub-field Non-Repeatable.*

   Example: 440--Title-Only Series Note (Traced Same).

   Form: $aSeries Title, $vVolume,Volume,Volume,... *or* inclusive numbers, e.g., $vv. 11-15.

3. *Field Non-Repeatable, Sub-field Repeatable.*

   Example: 260--Imprint.

   Form: $aPlace$bPublisher$cDate;$aPlace$bPublisher-$cDate.
   (Dates non-contiguous)

4. Field Non-Repeatable, Sub-field Non-Repeatable.

Example: 260--Imprint.

Form: $aPlace$aPlace$bPublisher$cDate,Date.
(Dates contiguous; delimiter not required)

Multiple Element Field, Single Value.

5. Field Repeatable, Sub-field Non-Repeatable.

Example: 051--LC Copy Statement.

Form: $aClass No.$bBook No.$cCopy Information.

6. Field Non-Repeatable, Sub-Field Non-Repeatable.

Example: Tag 041--Languages.

Form: $aLanguage Codes of Text$bLanguage Codes of Summaries

Single Element Field, Multiple Values.

7. Field Repeatable, Sub-field Non-Repeatable.

Example: 490--Series Untraced or Traced Differently.

Form: 490$aXXXX; XXXX.

8. Field Non-Repeatable, Sub-field Repeatable.

Example: 082--Dewey No.

Form: $aXXX$aXXX$a...

Single Element Field, Single Value.

9. Field Repeatable, Sub-field Non-Repeatable.

Example: 500--General Notes.

Form: $aXXXXXXXX.

10. Field Non-Repeatable, Sub-field Non-Repeatable.

Example: 241--Romanized Title.

Form: $aXXXXXXXX.

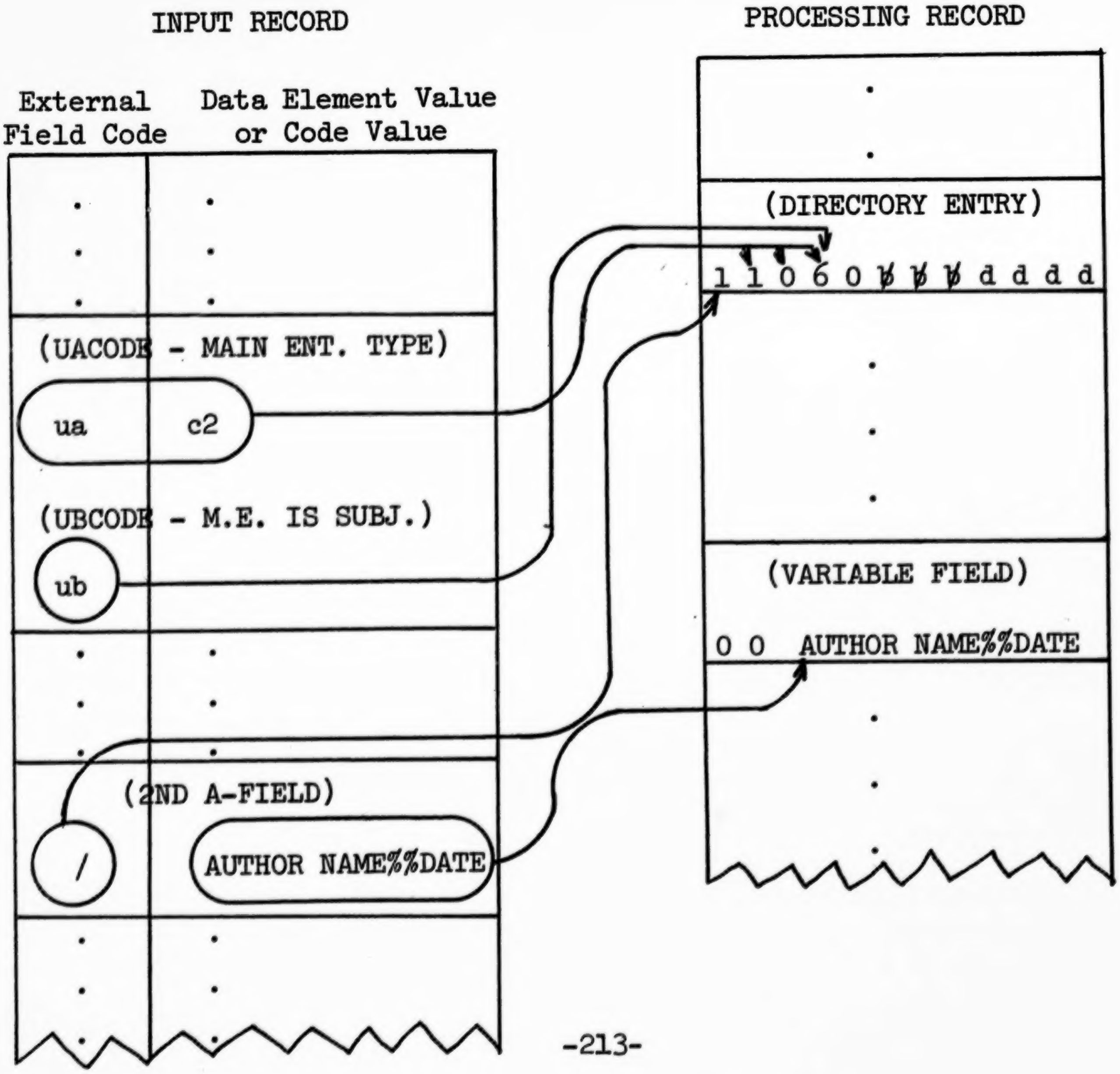
2. Mapping of Input Codes. To illustrate more precisely how the input format works, it must be remembered that the input codes do not necessarily bear a one-to-one correspondence to the field codes in the Processing Record. The external fields generally speaking map into logically identical internal fields, sometimes on a many-to-one basis (e.g., the place, publisher, and date elements are separate fields in the input record but map into one internal field with three sub-fields in the revised MARC II format).

Notationally their codes are different, usually in a one-to-many relation, e.g., the second A-Field slash, with no "ua" code, translates to Tag 100, Indicator value of 1=Main Entry, Personal Name, Single Surname sub-type, Condition: main entry is not subject. The mapping is complex, that is, input fields and codes are translated (expanded), re-arranged, concatenated, or split up to form the internal fields. Fig. 3 is an illustration of such a mapping.

There are four levels of coding in the tag and indicator of a MARC record. These encompass the aspects of each data entity needed to serve the four functions mentioned above. In some cases, the coding has been extended by the device of "stacked" coding, to serve auxiliary functions such as recording the fact that the main entry heading is identical with the subject of the book. This allows the user the option of printing out the record under the name as a subject heading in a divided catalog.

FIGURE 3:

EXAMPLE OF INPUT FORMAT MAPPING INTO PROCESSING FORMAT


INPUT RECORD
PROCESSING RECORD
External Field Code
Data Element Value or Code Value
(UACODE - MAIN ENT. TYPE)
ua
c2
(UBCODE - M.E. IS SUBJ.)
ub
(2ND A-FIELD)
/
AUTHOR NAME%%DATE
(DIRECTORY ENTRY)
1 1 0 6 0 ȸ ȸ ȸ d d d d
(VARIABLE FIELD)
0 0 AUTHOR NAME%%DATE

In Fig. 3, the four levels are exemplified as follows:

	MARC II Code Type
Level I - Function of the Field identified by the code	TAG (digit 1)
e.g., 2nd A-Field slash = Main Entry Heading	$\boxed{1}$ XX
Level II - Type of Entity in a Heading (ua)	TAG (digits 2-3)
e.g., c = Corporate Body	1 $\boxed{10}$
Level III - Sub-Type of Entity	(Indicator 1)
e.g., 2 = Corporate Body in form of Direct Order and Role Relation is Main Entry is Not Subject ("stacked code")	110 $\boxed{2}$
Level IV - Role Relation of Functional Entity	(Indicator 1)
e.g., "ub" code in input stream resets Role Relation in stacked code above to 6 = Main Entry is Subject	110 $\boxed{6}$

It should be noted that there is no necessary dependence between coding in the input format and the processing format. The internal code could, for example, be stored in two separate indicators, e.g., Indicators 1 and 2, with no effect on the input format except to change a table in the input edit program.

As part of this translation process, the edit program must perform all the bookkeeping actions necessary to assemble the record in Processing Format (e.g., placing tags in directory entries, counting field lengths, etc.). Redundant data or purely intermediary codes are purged. Default settings and initializations are completed. Computed values are set, either from the input data (e.g., illustration codes in the Fixed Fields) or by logical combinations (e.g., a blank 2nd slash with the absence of a "ua" code means a title main entry is being input). Character translation (dependent upon the particular keyboard device used) is made into the representation of the computer.

Some other differences between the input format and the processing format are:

Repeatability. Certain codes in the input record, e.g., the 10 A-Field slashes, are not repeatable, whereas their corresponding internal codes are repeatable.

Presence. Certain Fixed Length Data Elements are input only when present, whereas the internal record always contains each of these elements, even if the value is "off", "none", or such.

Level. Certain codes in the input format may translate to a tag in one case and a sub-field in another case, e.g., the 3rd A-Field slash translates to a tag (Title); the fourth slash translates to a sub-field delimiter (remainder of title).

C. RECORD COMPOSITION

1. Logical Structure. A complete input record is composed of a stream of data and codes, in three segments: The Indicator segment (I-Fields), the Body of the Catalog Card (A-Fields), and the Remaining Data (B-Fields). Each segment is composed of several fields. The fields are of either fixed or variable-length type, depending on content. The data elements are variable or fixed in presence. The codes are correspondingly fixed or variable in presence. The codes to the left and below the catalog card box on the coding sheet are the I-Fields, in pre-printed form.

The coding sheet* designed for experimental use in the File Organization project provides space to record certain information which is not explicitly recorded on the catalog card, or for combining in one field the information (e.g., on holdings) gathered from several cards representing the same title, or to record information in coded rather than natural form. The I-Fields are structured in two ways: either checklist or fill-in. The codes are pre-printed on the sheet in checklist format as a memory aid to the editor. The boxes are checked to indicate that they should be keyboarded. Codes which are merely checked are of the self-transmitting type - the information they carry is embodied in a first letter defining the general group of the code; the second letter is a specific value for the code. In some cases there are multiple values assumed by a particular internal data element (e.g., Content Form may be checked from 0-4 boxes). The second letter values will be inserted into the proper element in the Fixed Length Data Elements Field in the Processing Format.

Where the list of options for a code is too lengthy to be pre-printed on the coding sheet, the editor selects a code from a reference list in the coding manual and writes in the proper values in the box. In Type of Main Entry, "ua" merely identifies the value which it transmits. An entry could be "uac@" meaning "Type of Main Entry = Corporate Name, Direct Order, Main Entry in this record is not subject of the work".

---

*See Fig. 11, Part III, Final Report of File Organization Project, Phase I.

Certain data elements are not explicitly coded: e.g., the Master Record Number is always placed first in the input record and is recognized by position.

The 1st section of the coding sheet is that outside of the catalog card image. It contains boxes:

- to provide values to certain elements in the Record Leader.
- to transmit codes to elements in the Fixed Length Data Elements Field in the Processing Record.
- to provide values for parts of tags and indicators that are not supplied by coding in the other segments of the input record.
- to supply data that cannot be recorded on the compact face of the catalog card image, e.g., call numbers from other campuses possessing copies of the work.
- to record certain control information, such as editing and keyboarding statistics.

The codes are always 2 or 4 character combinations, both to facilitate expansion and for validity checking.

In the card image portion there are two types of coding:

A-fields. The body of the source record, the ten fields or data elements most frequently occurring on the card.

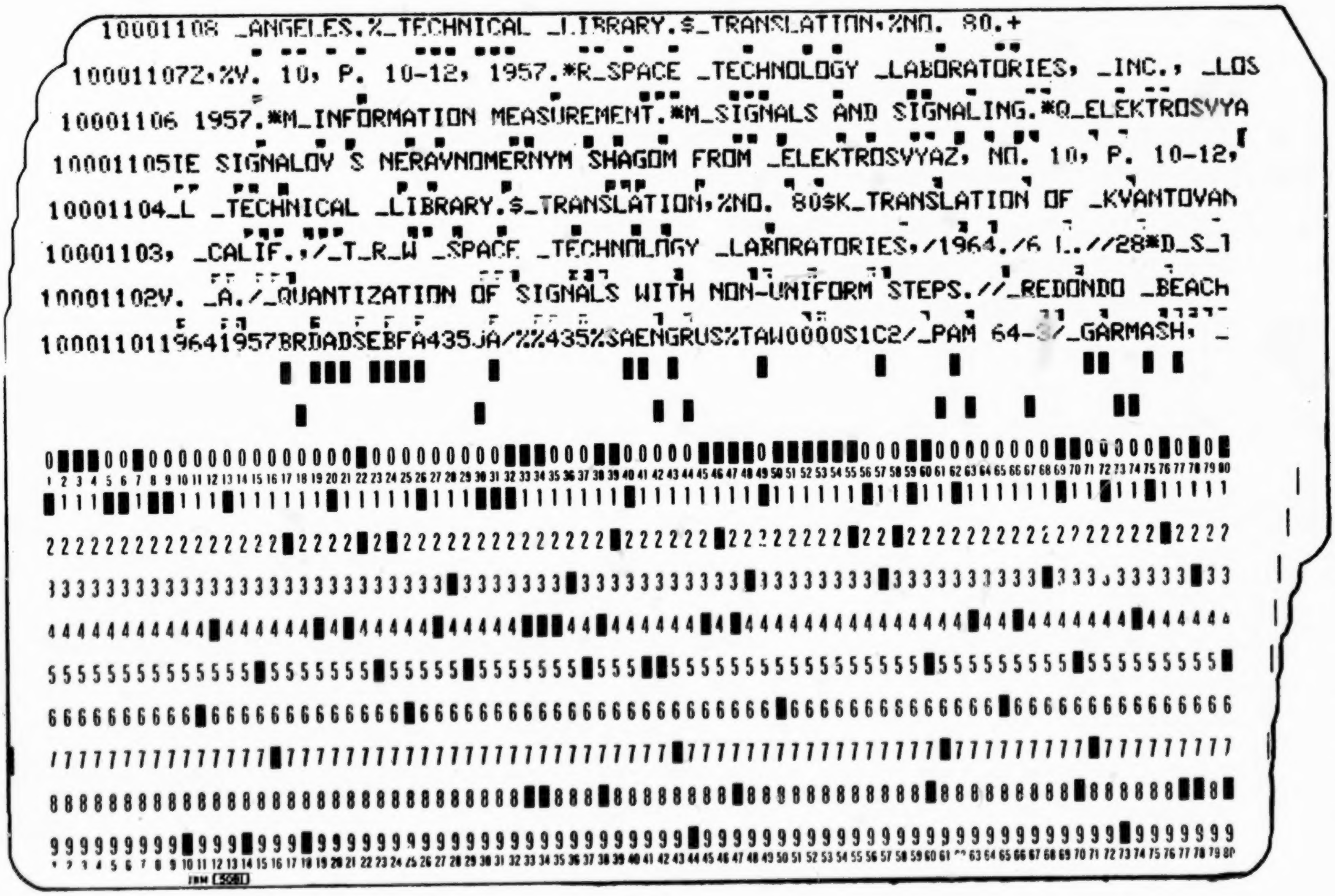
B-fields. Collateral data which occurs with less predictability on the cards, such as supplementary notes.

The tables in Section D list the data element names and their codes in the ILR input format, together with the corresponding Processing Format codes into which they map.

2. Physical Record. The input record is composed of a variable number of characters placed in contiguous locations on an input medium, e.g., punched cards. A physical input record is composed of one unit of the medium used by the keyboarding device: e.g., a tab card. There might be several tab cards in a complete logical record. (See Fig. 4.)

a. Beginning of record. A master record number is placed at the beginning of the input record. For the purposes of the present conversion, it is 6 characters in length.

b. Continuation segments. If a medium such as tab cards is used, the record number is repeated in character positions 1-6 in each succeeding card. A serial card number is placed in c.p. 7-8, right justified, beginning with "01". A maximum of 15 cards per decklet is currently permitted in the INFOCAL program.



```
10001108 _ANGELES.%_TECHNICAL _LIBRARY.$_TRANSLATION,%NO. 80.+
10001107Z,%V. 10, P. 10-12, 1957.*R_SPACE _TECHNOLOGY _LABORATORIES, _INC., _LOS
10001106 1957.*M_INFORMATION MEASUREMENT.*M_SIGNALS AND SIGNALING.*Q_ELEKTROSVYA
10001105IE SIGNALOV S NERAVNOMERNYM SHAGOM FROM _ELEKTROSVYAZ, NO. 10, P. 10-12,
10001104_L _TECHNICAL _LIBRARY.$_TRANSLATION,%NO. 80$K_TRANSLATION OF _KVANTOVAN
10001103, _CALIF.,/_T_R_W _SPACE _TECHNOLOGY _LABORATORIES,/1964./6 L.//28*D_S_1
10001102V. _A./_QUANTIZATION OF SIGNALS WITH NON-UNIFORM STEPS.//_REDONDO _BEACH
1000110119641957BRDADSEBFA435JA/%%435%SAENGRUS%TAW0000S1C2/_PAM 64-3/_GARMASH, _
```

FIGURE 4:

EXAMPLE OF TAB CARD DECKLET - ILR INPUT FORMAT

c. End of record. An explicit symbol is used to signify the end of the logical record. Currently, a "+" is used for this purpose.

2. Logical Record. The logical record content and encoding is organized to be independent of the physical medium used for input. (It is "free-floating".) It is not dependent on the symbols provided on any particular device, although a particular implementation of the input format implies selection of a particular set of symbols for a given device. The set chosen in the remainder of this specification is based on the keyboard of a Standard 029 keypunch.

D. DATA ELEMENTS AND CODES

The following tables list the complete data elements and codes currently implemented in the ILR Input Record Format.

FIGURE 5:

ILR INPUT RECORD FORMAT

I-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS.; SUB-FLD. CODE IF IN "()"
1. Master Record Number (ILR)	None		L			40-46
2. Record Status:			L			12
Uncertified new record	None					(0)
Certified new record	"					(1)
Changed record	a	a				(2)
Deleted record	None					(3)
3. Publication Date 1	None		F			2-5
4. Publication Date 2	"		F			6-9
5. Pub. Date Type:			F			1
2 dates, 2nd is copyrt.	b	c				(c)
2 dates, 2nd is terminal	b	m				(m)
Date not known	b	n				(n)
Digits missing	b	q				(q)
Prev. public., all digits	b	r				(r)
Single date, all digits	b	s				(s)
6. Type of Record:			L			14
Language materials (books)	c	a				(a)
Language materials (mss.)	c	b				(b)
Music, printed	c	c				(c)
Music, manuscript	c	d				(d)
Maps/atlases, printed	c	e				(e)

KEY to components: L = Leader, D = Directory, F = Fixed Length Data Elements Field, V = Variable Fields

FIGURE 5 (Cont.):

ILR INPUT RECORD FORMAT

I-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS.; SUB-FLD. CODE IF IN "()"
6. Type of Record (Cont.):			L			14
Maps, manuscript	c	f				(f)
Motion pic. & filmstrips	c	g				(g)
Microform publications	c	h				(h)
Phonorecords, spoken	c	i				(i)
Phonorecords, music	c	j				(j)
Pictures, designs, etc.	c	k				(k)
Machine-readable data	c	l				(l)
7. Bibliographic Level:			L			15-17
Analytical	d	a				(a)
Collective	d	c				(c)
Monograph	d	m				(m)
Serial	d	s				(s)
8. Cataloging Source Type:			L			33
Central (LC Card)	e	a				(a)
Local original	e	b				(b)
NUC	e	c				(c)
Other	e	d				(d)
9. Book Not in LC (LC Call No. Bracketed)	e	z	D	050	1	
10. Agency Code for Cataloging Source	f	a	L			34-36
11. Agency Code for Adaptor of Catalog Card	f	b	L			37-39

FIGURE 5 (Cont.):
ILR INPUT RECORD FORMAT

I-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS.; SUB-FLD. CODE IF IN "()"
12. Form of Micro-reproduction:			F			10
Microfilm	g	a				(a)
Microfiche	g	b				(b)
Micro-opaque	g	c				(c)
13. Content Forms:			F			11-14
Bibliographies	h	a				(a)
Catalogs	h	b				(b)
Indexes	h	c				(c)
Abstracts	h	d				(d)
Dictionaries	h	e				(e)
Encyclopedias	h	f				(f)
Directories	h	g				(g)
Yearbooks	h	h				(h)
Statistics	h	i				(i)
Handbooks	h	j				(j)
Other	h	k				(k)
14. Holdings:	j	a-j	D	090		
Call Number	None		V			None
Copy Number	%		V			(%)
Library Code/Location	%		V			(%)
Total copies in location	%		V			(%)
15. Government Publication:			F			15
U.S. Federal	k	a				(a)
California State	k	b				(b)

FIGURE 5 (Cont.):

ILR INPUT RECORD FORMAT

I-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS.; SUB-FLD. CODE IF IN "()"
15. Government Publication (Cont.):			F			15
California County/Muni.	k	c				(c)
International Orgn.	k	d				(d)
Other Countries	k	e				(e)
16. Conference Publication	m	a	F			16 (1)
17. Main Entry Heading Is Repeated in Body of Card	n	a	F			17 (1)
18. Literary Group Filing Indicator:			F			18
Complete/collected works	p	a				(a)
Selected works	p	b				(b)
Prolific author	p	c				(c)
19. Cancel Title Added Entry in Dictionary Cat. Only (See AA 33P3 and 6)	q	a	F			19 (1)
20. Cancel Title Added Entry in Both Dict. & Divided Catalogs	r	a	D	240	0	
21. Language codes:	s	a	D	003	X[+]	
Lang. of text sub-field	None		V			None
Summaries	%		V			(%)
22. Translation Indicator	t	a	D	003	1	

[+] An "X" means this portion of the code is supplied via the input codes in another element of this table, or another table (q.v.).

FIGURE 5 (Cont.):

ILR INPUT RECORD FORMAT

I-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS.; SUB-FLD. CODE IF IN "()"
23. Type of Main Entry Code (See Fig. 8 for table of values which complete the tag and indicator)	u	a	D	1XX	X	
24. Main Entry is Subject Indicator (See Fig. 8 for table of values for Indicator 1, for which this is a stacked code)	u	b	D	1XX	X	
25. Main Entry is Publisher	u	c	D	261	1	
26. Type of Added Entry Codes (See Fig.9-12 for table of values for each category of added entry):	w	2-char. repeatable codes				
Series Traced Same			D	4XX	X+	
Subject Added Entries			D	6XX	X	
Other Added Entries			D	7XX	X	
Series Traced Diff.			D	8XX	X	

+ An "X" means the 2-character codes in the Value column translate into the 2nd and 3rd character of the tag and the value for Indicator 1.

FIGURE 6:
ILR INPUT RECORD FORMAT

A-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS; SUB-FLD. CODE IF IN "()"
1. Local Call Number	/		D	090		
2. Main Entry Heading:	/		D	1XX	X+	
Personal Name						
Name sub-field	None		V			None
Titles of honor, etc.	%		V			(%)
Identifier	%		V			(%)
Relator	%		V			(%)
Corporate Name (&Firm Name)						
Name sub-field	None		V			None
Each subheading unit	%		V			(%)
Conference Name						
Name sub-field	None		V			None
Number	%		V			(%)
Place	%		V			(%)
Date of conf.	%		V			(%)
Corporate with Form Subhd.						
Name sub-field	None		V			None
Form subheading	%		V			(%)
Uniform Title M.E. Heading						
Title sub-field	None		V			None
Other information	"		V			"

+An "X" means this portion of the code is supplied via the input codes described in another table in this specification (q.v.). Each variable field contains 2 positions for Indicators 6 and 7 at the beginning of the field. The data starts in character position 3.

FIGURE 6 (Cont.):
ILR INPUT RECORD FORMAT

A-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS; SUB-FLD. CODE IF IN "()"
3. Supplied Title (Uniform Title in Interposed Position):						
Not printed on LC cards	$$+					
Printed on LC cards	$		D	200	1	
4. Title Statement:						
Short Title	/		D	240	X	None
Remainder of Title	/		V			(%)
Remainder of Title Page Transcription	%		V			(%)
5. Edition Statement:						
Edition	#		D	250		None
Remainder of Statement	%		V			(%)
6. Place of Publication	/	R($)++	D	260		
7. Publisher	/	R($)	D	261	X	
8. Date of Publication	/		D	262		
9. Collation Statement:						
Pagination/Volumes	/		D	300		None
Illustrative Matter	/		V			(%)
Size	/		V			(%)
10. Bibliographic Price	$		D	350		R(%)++

+This code does not exist in the present version of INFOCAL.

++An "R" means the data element is repeatable. If tag is non-repeatable, the 2nd and succeeding values of the element occurring in a record are assigned a preceding delimiter ("%"). If tag is repeatable, "$" translates to tag.

FIGURE 7:
ILR INPUT RECORD FORMAT

B-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME (Primary Series of Codes)	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS; SUB-FLD. CODE IF IN "()"
1. Series Note, Traced Same (Auth. +Title form):	*	a	D	4XX	X	
Personal Name						
Sub-fields are the same as in Main Entry Heading, with addition of:						
Series title sub-field	$		D	408		None
Volume/Number	%		V	"		(%)
Corporate Name (& Firm Name)						
Sub-fields are the same as in Main Entry Heading, with addition of:						
Series title sub-field	$		D	418		None
Volume/Number	%		V	"		(%)
Conference Name						
Sub-fields are the same as in Main Entry Heading, with addition of:						
Series title sub-field	$		D	418		None
Volume/Number	%		V			(%)
See Fig. 9 for codes which identify remainder of tag digits + Indicator.						
2. Series Note, Traced Same (Title only)	*	b	D	440		
3. Series Note, Not Traced	*	c	D	490	0	
4. Series Note, Traced Differently	*	d	D	490	1	
5. National Bibliography Number	*	e	D	011		
6. Bibliography Note	*	f	D	500		
7. "IN" Analytic Note	*	g	D	570		
8. Dissertation Note	*	h	D	510		

FIGURE 7 (Cont.):
ILR INPUT RECORD FORMAT

B-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS; SUB-FLD. CODE IF IN "()"
9. Contents Note (Formatted)	*	i	D	520		
10. "Bound With" Note	*	j	D	530		
11. General Notes	*	k	D	550		
12. Dash Supplements	*	ℓ+				
13. LC Subject Headings:	*	m	D	6XX	X	
Personal Name						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	608		
Corporate Name (& Firm Name)						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	618		
Conference Name						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	618		
Corporate with Form Subhd.						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	628		
Uniform Title Heading						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	638		
See Fig.10 for codes which identify other types of LC Subject Hdgs.						

+This code does not exist in the present version of INFOCAL.

FIGURE 7 (Cont.):
ILR INPUT RECORD FORMAT

B-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS; SUB-FLD. CODE IF IN "()"
14. LC Subject Subdivisions	--	(2 hyphens)	D	6XX		
15. Bibliographic History Note	*	p+	D	503		
16. Added Entries, Non-Subject & Non-Series:	*	q	D	7XX	X	
Personal Name						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	708		
Corporate Name (& Firm Name)						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	718		
Conference Name						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	718		
Corporate with Form Subhd.						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	728		
Uniform Title Heading						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Title of Book	$		D	738		
See Fig.11 for codes which identify other types of added entries of this category.						

+This code does not exist in the present version of INFOCAL. The Processing Format tags referred to are MARC II definitions.

FIGURE 7 (Cont.):
ILR INPUT RECORD FORMAT

B-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS.; SUB-FLD. CODE IF IN "()"
17. Series Added Entries (Traced Differently):	*	r	D	8XX	X	
Personal Name						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Series title sub-field	$		D	808		None
Volume/Number	%		V	"		(%)
Corporate Name (& Firm Name)						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Series title sub-field	$		D	818		None
Volume/Number	%		V	"		(%)
Conference Name						
Sub-fields are the same as in Main Entry Heading, with the addition of:						
Series title sub-field	$		D	818		None
Volume/Number	%		V	"		(%)
See also Fig. 12 for Series Added Entry in Title-Only Form.						
18. Title Romanized Note (No Added Entry)	*	n	D	210	0	
19. LC Call Number:	*	s	D	050	0	
Class Number sub-field	None		V			None
Book Number	"		V			"

FIGURE 7 (Cont.):
ILR INPUT RECORD FORMAT

B-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS; SUB-FLD. CODE IF IN "()"
20. LC Copy Number Statement:	*	t	D	051		
Class Number sub-field	None		V			None
Book Number	"		V			"
Copy Information	%		V			(%)
21. Local Library Copy Statement:	*	u	D	091		
Class Number sub-field	None		V			None
Book Number	"		V			"
Copy Information	%		V			(%)
22. Dewey Decimal Class No.:	*	w	D	030		R(%)
Segment sub-field	/		V			R(/)
23. LC Card Number	*	x	D	001		
24. Overseas Acquisition No.	*	y	D	013		

[End of primary series]

FIGURE 7 (Cont.):
ILR INPUT RECORD FORMAT

B-FIELDS: DATA ELEMENTS AND CODES			PROCESSING FORMAT CODES			
ELEMENT NAME (Secondary Series of Codes)	FIELD CODE	CODE VALUE(S)	COMPONENT	TAG	INDICATOR 1 HAS VALUE	CHAR. POS.; SUB-FLD. CODE IF IN "()"
1. Standard Book Number	!	a+	D	012		
2. "Limited Use" Note	!	b	D	540		
3. Abstract or Annotation	!	c	D	560		
4. [Reserved for later use]	!	d+				
5. MeSH Main Headings	!	e	D	660		
6. NAL Agric./Biol. Vocab.	!	f+	D	670		
7. Local Subject Heading System	!	g+	D	690		
8. NLM Call Number	!	h	D	060		
9. NAM Call Number	!	i	D	070		
10. NAL Subj. Category No.	!	j+	D	071		
11. Cooperative Cataloging Library Call Number	!	k	D	052		
12. Special Classification System	!	ℓ+				
13. Supt. of Documents Catalog Number	!	m+				
14. Agency Name of Cooperative Cataloging Source	!	n	V	052		(%)
15. "Full Name" Notes	!	p	D	580		
16. Title Romanized Note (Make Added Entry)	!	q	D	210	1	
17. MeSH Topical Subheadings	!	t	D	661		
18. MeSH Geographic Subheadings	!	u	D	662		
19. MeSH Time Period Subheadings	!	w	D	663		
20. MeSH Form Subheadings	!	x	D	664		
21. NUC Card Number	!	y+				

[End of Secondary Series]

+This code does not exist in the present version of INFOCAL. The Processing Format tags referred to are MARC II definitions.

FIGURE 8:
INPUT CODE VALUES TABLE FOR TYPE OF MAIN ENTRY
(UA CODE)

TYPE OF ENTITY (Function = Main Entry Heading) Combined with: A-Field 2nd "/"	SUB-TYPE	CODE FOR TYPE	CODE FOR SUB-TYPE	PROCESSING FORMAT CODES		
				TAG	INDICATOR 1	
					Main Ent. Not Sub.	Main Ent. Is Sub.
PERSONAL NAME	Forename	p	0/4+	100	0	4
	Single Surname	p	1/5	100	1	5
	Multiple Surname	p	2/6	100	2	6
	Name of Family	p	3/7	100	3	7
CORPORATE NAME	Surname (Inverted)	c	0/4	100	0	4
	Place or Pl. + Name	c	1/5	110	1	5
	Name (Direct Order)	c	2/6	110	2	6
CONFERENCE NAME	Surname (Inverted)	e	0/4	111	0	4
	Place or Pl. + Name	e	1/5	111	1	5
	Name (Direct Order)	e	2/6	111	2	6
FIRM NAME		f	0/1	112	0	1
CORPORATE NAME WITH FORM SUBHD.		r	0/1	120	0	1
UNIFORM TITLE	General	u	0/1	130	0	1
	Anonymous Classic	a	0/1	131	0	1
TITLE MAIN ENTRY		t	0	240	0	1

+One-digit codes to the right of the slash are set by input of the "ub" code in the I-Fields.

FIGURE 9:
INPUT CODE VALUES TABLE FOR TYPE OF ADDED ENTRIES
(W CODE)

TYPE OF ENTITY (Function = Series Notes, Traced Same, Author+ Title form) Combine with: B-field *a	SUB-TYPE	CODE FOR TYPE	CODE FOR SUB-TYPE	PROCESSING FORMAT CODES		
				TAG	INDICATOR 1	
					Auth. Not Main Ent.	Auth. In Main Ent
PERSONAL NAME	Forename	p	0/4+	400	0	4
	Single Surname	p	1/5	400	1	5
	Multiple Surname	p	2/6	400	2	6
	Name of Family	p	3/7	400	3	7
CORPORATE NAME	Surname (Inverted)	c	0/4	410	0	4
	Place or Pl. + Name	c	1/5	410	1	5
	Name (Direct Order)	c	2/6	410	2	6
CONFERENCE NAME	Surname (Inverted)	e	0/4	411	0	4
	Place or Pl. + Name	e	1/5	411	1	5
	Name (Direct Order)	e	2/6	411	2	6
FIRM NAME		f	0/1	412	0	1

+One-digit codes to the right of the slash are set by input of the "uc" code in the I-Fields.

FIGURE 10:
INPUT CODE VALUES TABLE FOR TYPE OF ADDED ENTRIES (W CODE)

TYPE OF ENTITY (Function = Subject Added Entries & Subj. Subdivisions) Combine with: B-Field "*m"	SUB-TYPE	CODE FOR TYPE	CODE FOR SUB-TYPE	PROCESSING FORMAT CODES	
				TAG	INDICATOR 1
PERSONAL NAME	Forename	p	0	600	0
	Single Surname	p	1	600	1
	Multiple Surname	p	2	600	2
	Name of Family	p	3	600	3
CORPORATE NAME	Surname (Inverted)	c	0	610	0
	Place or Pl. + Name	c	1	610	1
	Name (Direct Order)	c	2	610	2
CONFERENCE NAME	Surname (Inverted)	e	0	611	0
	Place or Pl. + Name	e	1	611	1
	Name (Direct Order)	e	2	611	2
FIRM NAME		f	0	612	None
CORPORATE NAME WITH FORM SUBHD.		r	0	620	None
UNIFORM TITLE	General	u	0	630	None
	Anonymous Classic	a	0	631	None
TITLE OF WORK		t	0	640	None
TOPICAL		0	0	650	None
GEOGRAPHIC NAME		1	0	651	None
POLIT. JURISDICT.		2	0	652	None

FIGURE 10 (Cont.): INPUT CODE VALUES TABLE FOR TYPE OF ADDED ENTRIES (W CODE)

TYPE OF ENTITY (Function = Subject Added Entries) Cont'd.	SUB-TYPE	CODE FOR TYPE	CODE FOR SUB-TYPE	PROCESSING FORMAT CODES	
				TAG	INDICATOR 1
NAMES NOT CAPABLE OF AUTH.		3	0	653	None
GENERAL SUBJ.+ SUBDIVISIONS		5	0	655	None
PERIOD SUBJ.+ SUBDIVISIONS		6	0	656	None
PLACE SUBJ.+ SUBDIVISIONS		7	0	657	None

+See also B-Fields, Primary Codes, item No. 14.

FIGURE 11:
INPUT CODE VALUES TABLE FOR TYPE OF ADDED ENTRIES
(W CODE)

TYPE OF ENTITY (Function = Non-subject, Non-series Added entries) Combine with: B-Field "*q"	SUB-TYPE	CODE FOR TYPE	CODE FOR SUB-TYPE	PROCESSING FORMAT CODES			
				TAG	INDICATOR 1		
					Role		
					Alt.	Conn.	Analyt.
PERSONAL NAME	Forename	p	@/d/h	700	@	d	h
	Single Surname	p	a/e/i	700	a	e	i
	Multiple Surname	p	b/f/j	700	b	f	j
	Name of Family	p	c/g/k	700	c	g	k
CORPORATE NAME	Surname (Inverted)	c	@/d/h	710	@	d	h
	Place or Pl. + Name	c	a/e/i	710	a	e	i
	Name (Direct Order)	c	b/f/j	710	b	f	j
CONFERENCE NAME	Surname (Inverted)	e	@/d/h	711	@	d	h
	Place or Pl. + Name	e	a/e/i	711	a	e	i
	Name (Direct Order)	e	b/f/j	711	b	f	j
FIRM NAME		f	0/1/2	712	0	1	2
CORPORATE NAME WITH FORM SUBHD.		r	0/1/2	720	0	1	2
UNIFORM TITLE TRACING	General	u	0/1/2	730	0	1	2
	Anonymous Classic	u	0/1/2	731	0	1	2
TITLE ADDED ENTRIES (TRACED DIFFERENTLY FROM SHORT TITLE)	Partial Title - Work in Hand	t	0	740	0		
	Title Added Ent.- Another Work	t	1	740		1	
	Analytic Title	t	2	740			2
NAME NOT CAPABLE OF AUTHORSHIP		n	0/1/2	753	0	1	2

FIGURE 12:
INPUT CODE VALUES TABLE FOR TYPE OF ADDED ENTRIES
(W CODE)

TYPE OF ENTITY (Function = Series Added Entry, Traced Differently) Combine with: B-Field "*r"	SUB-TYPE	CODE FOR TYPE	CODE FOR SUB-TYPE	PROCESSING FORMAT CODES	
				TAG	INDICATOR 1
PERSONAL NAME	Forename	p	0	800	0
	Single Surname	p	1	800	1
	Multiple Surname	p	2	800	2
	Name of Family	p	3	800	3
CORPORATE NAME	Surname (Inverted)	c	0	810	0
	Place or Pl. + Name	c	1	810	1
	Name (Direct Order)	c	2	810	2
CONFERENCE NAME	Surname (Inverted)	e	0	811	0
	Place or Pl. + Name	e	1	811	1
	Name (Direct Order)	e	2	811	2
FIRM NAME		f	0	812	None
TITLE-ONLY FORM		t	0	840	None

E. PRESENCE AND DEFAULT CONDITIONS

For purposes of the logic of the computer edit program, the I-Field elements have special "presence" conditions. Fig. 13 lists the status of each input field code according to its presence. Note that the absence of an input field does not necessarily imply that the data element will be absent from the processing record. For those I-fields always present, it was decided to define default settings that could be made by program without any action by human editor or keyboard operator. These experimental default coding conditions are listed in Figs. 14-15.

For purposes of setting Indicator 2, the value of which records the sequence numbers of fields with multiple occurrences in a record, Fig. 16 shows the fields which are currently repeatable in the INFOCAL program.

FIGURE 13:
PRESENCE OF FIELDS IN AN INPUT RECORD (MANUAL EDIT)

PROGRAM SYMBOL	RECORD COMPONENT EXTERNAL CODE	DATA ELEMENT	ALWAYS PRESENT
LISTATUS	I-Fields "a"	Record Status	Only present to override default setting
LIDTYPE	I-Fields "b"	Date of Publication Type	Only present to override default setting
LITYPE	I-Fields "c"	Record Type	Only present to override default setting
LIDATE1	I-Fields - no external code	Date of Pub. 1	No
LIDATE2	I-Fields - no external code	Date of Pub. 2	No
LIBLEVEL	I-Fields "d"	Bibliographic Level	Only present to override default setting
LISOURCE	I-Fields "e"	Source Type of Catalog Card	Only present to override default setting
LIORIGIN	I-Fields "f"	Agency Code of Origin of Mach. Rec.	Only present to override default setting
LIPROSOR	I-Fields "f"	Agency Code of Processor of Mach. Rec.	Only present to override default setting
LIMICROR	I-Fields "g"	Form of Microreproduction	No
LIFORM	I-Fields "h"	Content Form(s)	No
JCODE	I-Fields "j"	Holdings Field(s)	Yes (part set by default)
LIGOVPUB	I-Fields "k"	Govt. Pub. Indic.	No
LICONPUB	I-Fields "m"	Conference Pub. Indicator	No
LIMEBODY	I-Fields "n"	Main Entry in Body	No

FIGURE 13 (Cont.):
PRESENCE OF FIELDS IN AN INPUT RECORD (MANUAL EDIT)

PROGRAM SYMBOL	RECORD COMPONENT EXTERNAL CODE	DATA ELEMENT	ALWAYS PRESENT
LILITGRP	I-Fields "p"	Literary Group Indic.	No
LICNCELT	I-Fields "q"	Cancel Title Added Entry - Dict.	No
RCODE	I-Fields "r"	Cancel Title Added Ent.-Dict. & Div.	No
SCODE	I-Fields "s"	Languages Codes	Only present to override default setting
TCODE	I-Fields "t"	Translation Indicator	No
UACODE	I-Fields "u"	Type of Main Entry	Only present to override default setting
UBCODE	I-Fields "u"	Main Entry Is Subj. Indic.	No
UCCODE	I-Fields "u"	Main Entry is Publ'r. Indic.	No
WCODE	I-Fields "w"	Type of Added Entries Codes	No

FIGURE 14:
INFOCAL DEFAULT INITIALIZATIONS REFLECTED IN PROCESSING RECORD FORMAT WHEN NO INPUT CODES ARE RECEIVED

PROGRAM SYMBOL	DATA ELEMENT	VALUE(S)
LIPREFX	LEADER	
LISDATE	Record Status Date	6 blanks
LISTATUS	Record Status	EBCDIC zero = uncertified
LILEGCNT	Legend Extension	EBCDIC zero = no extension
LITYPE	Record Type	a = language material, printed
LIBLEVEL	Bibliographic Level	mb̸b̸ = monograph
LIINDCNT	Indicator Count	EBCDIC 5 = no. of indicator bytes in a directory entry
LIORIGIN	Agency Code for Origin of Machine Record	003 = original input at ILR 790 = original input at UCSC
LIPDATE	Date of Processing of Machine Record	6 blanks (Refers to date of translation of UCSC original format to ILR processing format)
LIPROSOR	Agency Code for Processor of Machine Record	003 = format translation done by ILR from UCSC original format
LIPSOURCE	Source Type of Catalog Card	a = central (LC Card)
LIAGENCY	Agency Code of Source Card	000 = LC
LIADAPTR	Agency Code of Adaptor of Source Card	3 blanks
LIFIXED	FIXED LENGTH DATA ELEMENTS FIELD	
	All elements are set to blanks except:	
LIDTYPE	Date of Publication Type	s = single date
LICONPUB	Conference Publication Indicator	0 = no
LIMEBODY	Main Entry in Body Indicator	0 = no
LICNCLT	Cancel Title Added Entry	0 = no (don't cancel title added entry--make one from short title for dict. cat.)

FIGURE 14 (Cont.):

INFOCAL DEFAULT INITIALIZATIONS REFLECTED IN PROCESSING RECORD FORMAT WHEN NO INPUT CODES ARE RECEIVED

PROGRAM SYMBOL	DATA ELEMENT	VALUE(S)
	ELEMENTS FROM I-FIELDS WHICH ARE TRANSMITTED TO VARIABLE DATA FIELDS	
JCODE	Holdings: Copy No. at This Location (2d sub-field)	01 = Copy No. at the shelf location indicated is Copy 1.
	Holdings: Shelf Location (3rd sub-field)	x = Stacks
	Holdings: Total Copies at This Location (4th sub-field)	01 = There is one copy assigned to this shelf location, with the copy no. shown in the second sub-field.
SCODE	Language Codes	eng = English is the language of the text.

FIGURE 15:

DEFAULT SETTINGS FOR INDICATOR 1 IN VARIABLE FIELD DIRECTORY ENTRIES TO WHICH INDICATOR IS APPLICABLE

INPUT COMPONENT AND CODE	DATA ELEMENT	ATTRIBUTE OF INDICATOR	CODE VALUE
I-Field sa	Languages	Single or Multi-language	0
B-Field *s	LC Call Number	Book in LC	0
A-Field /2 I-Field ua I-Field ub	Main Entry	Personal Name, Single Surname, Not Subject	1
A-Field $	Uniform Title	Printed on LC Cards	1
A-Field /3 I-Field ra	Title Statement	Make title added entry	1
A-Field /6 I-Field uc	Publisher	Publisher Not Main Entry	0

These codes are set programmatically in the absence of an external input code.

FIGURE 16:

LIST OF TAG NUMBERS WHICH ARE CURRENTLY REPEATABLE IN PROCESSING FORMAT

TAG NUMBER	FIELD NAME
051	LC Copy statement
090	Holdings
091	Local Library Copy statement
260	Place of Publication
261	Publisher
400	Series Notes
550	General Notes
6XX	Subject Added Entries
7XX	Other Added Entries
8XX	Series Added Entries

F. SUGGESTIONS FOR REVISION OF THE INPUT RECORD

1. General. The revised input format is predicated on a number of factors:

a. Unavailability of human editing talent in sufficient numbers to accomplish large scale conversion of bibliographic files.

b. Catalog cards follow a number of long-standing notational and stylistic conventions, e.g., Arabic numerals preceding subject tracings, series tracings enclosed in parentheses. These conventions will provide the textual clues in a bibliographic string that will support computer algorithms to accomplish a reasonably high degree of reliable field editing.

c. Ability of keyboard device operators to perform a minimum of obvious coding, e.g., keying a symbol which stands for a paragraph indention revealed by simple inspection of the catalog source card.

d. Sufficient regularities in catalog data, by form of material (e.g., monographs) and by language (e.g., English) that computer-assisted formatting will prove feasible. Large volumes of non-standard cards and idiosyncratic cataloging practices will tend to reduce the effectiveness of computer assigned codes.

e. The burden of proofreading and correction keyboarding must be less than in the straightforward system.

f. The conversion may have to be "phased", i.e., the Processing Format with full MARC II coding down to the level of every data element and indicator may not be achievable on the first pass of the conversion effort. Several iterations preceded by development and re-design may be necessary to achieve a fully-encoded machine file.

The following table (Fig. 17) is illustrative of the kinds of clues and patterns found on standard catalog cards which might be utilized in computer-assisted formatting. A total of six slash marks are used to break the card up into a set of segments. Various stylistic and notational clues are used to recognize by algorithm the fields within each segment. This is a proposed technique; more or less slashes may be required.

The recognition of sub-types of name and of the roles played by names in added entries, form perhaps the most difficult problem of tagging in MARC II. However, it is not unreasonable to suppose that a degree of accuracy could be attained that would be high enough so the total effort in proofing and correcting would be less than that now required to manually edit and proofread a record.

FIGURE 17:

REVISED FIELD CODING: A-FIELDS & B-FIELDS

ELEMENT NAME	MANUAL EDIT		COMPUTER EDIT Examples of Conditions Providing Clues to Algorithms
	Field Code	Recognize by	
Master Record Number (MRN)	None		Position in input string; length in digits
Local Call Number	None		Position following MRN
Main Entry Heading	/	Position;font	Punctuation; content patterns
Supplied Title (Interposed)	None		Brackets (left & right) must be present
Title Statement:			
Short Title	/	Paragraph indention	Right bracket of unif. title, if present
Remainder of Title	None		Punctuation; length in characters for title added entry purposes
Remainder of Title Page	None		Key word (e.g., "by")
Edition Statement	None		Key words; digits; etc.
Place of Publication	None		Punctuation
Publisher	None		Punctuation at end of place; authority table look-up; etc.
Date of Publication	None		Digits pattern
Collation Statement:			
Pagination	/	Paragraph indention	Punctuation; character patterns ("v.", "p.")
Illustration	None		Key words; set Fixed Field codes also
Size	None		Key word ("cm."); digits
Price	None		Key symbols; digits; punctuation
Series Notes (1st group)	None		Parentheses; sequence after collation or price

FIGURE 17 (Cont.):
REVISED FIELD CODING: A-FIELDS & B-FIELDS

ELEMENT NAME	MANUAL EDIT		COMPUTER EDIT Examples of Conditions Providing Clues to Algorithms
	Field Code	Recognize by	
Series Notes (2nd group) and all other notes	/	Paragraph indention	Key words & patterns (this is the most ambiguous region on a catalog card)
National Bibliography Number	None		Sequential position; key symbols; digits
Tracings: (code even if none present)	/	Paragraph indention;	
Beginning of subject tracings		Arabics & Roman Num.	Arabic numerals; match after conversion to a tagged authority file
Subject Subdivisions			Two hyphens (--)
Beginning of Other Added Entries			Roman numerals
Series Added Entries			Parentheses; key word; position
Fields at bottom of card:	/	Position	
LC Call Number			Character patterns
LC Copy Statement			Multiple hyphens ("dash"-on format)
Dewey Number			Digit pattern
LC Card Number			Digit pattern
Overseas Acquisition Number			Key symbols & abbrev.

2. An Illustrative Example. As an example of what can be done to design computer-assisted formatting, the following is a preliminary algorithm for delimiting the Short Title sub-field of a MARC II record. This procedure is a special case of string decomposition. This algorithm is phrased in such a form that it can be used either for manual editing or in a computer program. In essence, it is "machine-like" editing which is now performed manually by card preparation sections in libraries. It is of course subject to review by the cataloger on the basis of judgment as to what makes a title semantically informative and linguistically or grammatically acceptable. The algorithm is experimental and has been tested only in the manual coding effort undertaken during Phase I of the File Organization Project.

ALGORITHM FOR DELIMITING SHORT TITLE

Comment. It is assumed that the beginning of the title-page title statement will be explicitly identified in some way (visually, by a paragraph indention, or in the case of some title main entries, by hanging indention). The same will be true for the Collation field. An explicit field code such as "/" will precede the first word in the title. The problem is to determine the end of the Short Title, as defined in MARC II. The operational definition of Short Title is: that portion of the title statement which is to be used as a catalog entry heading or as a part of a heading, either main or added, and which might be used separately from the remainder of the catalog record, e.g., in index lists rather than in card catalogs. The short title will always be delimited when more than one of the MARC II sub-fields are present. This is independent of the Indicator setting for title added entry.

The algorithm does not identify the content of the next succeeding sub-field: that is assigned to a separate routine. This set of rules is still "fuzzy" in that cases of long titles in older catalog records which force the procedure to be carried through the string length cutoff step will not always produce a semantically or grammatically acceptable result.

A weakness in the procedure is its heavy dependence upon the punctuation used on the title pages of books and upon that inserted by catalogers according to conventional practice. Since this practice is rarely documented or consistently adhered to because of the variability of title pages and the need to rely on cataloger judgment, it is likely that the approach through punctuation is not susceptible of successive improvements.

Another weakness of the algorithm is that it cannot be consistently applied for situations where the beginning of the short title contains one or more words from the name of a real person (e.g., "Arthur Quiller-Couch; a biographical study of Q."). A title added entry would be more suitable if the subtitle were included, and so the delimiter is not placed after the semicolon.

The algorithm in draft form operates on single intervening marks of punctuation or other special symbols, multiple patterns of occurrence of a symbol, and combinations of symbols. A potential source of improvement in the algorithm would be a statistical study of frequency of these symbol patterns in catalog records, in order to optimize the sequence of the steps. At this stage the steps are presented in purely "intuitive" order.

FIGURE 18:

TABLE OF VALID SYMBOLS

GRAPHIC	NAME	EBCDIC HEX
[	Left Bracket	3E
]	Right Bracket	3F
.	Period	4B
<	Less Than Sign	4C
(	Left Parenthesis	4D
!	Exclamation Point	5A
)	Right Parenthesis	5D
;	Semi-colon	5E
,	Comma	6B
>	Greater Than	6E
?	Question Mark	6F
:	Colon	7A
"	Quotation Mark	7F

TABLE 1

UNCONDITIONAL (START)		Rule 1
1.	Scan left-right from start of	x
	Title paragraph to start of	
	Collation field (count)	
2.	Flag end of each sentence	x
	found (valid markers)*	
3.	For each intervening mark or symbol	x
	prior to first sentence marker, set	
	a position pointer (show type of	
	symbol)**	
4.	Go to Table 2	x

*A sentence is defined operationally as a phrase containing at least one character the end of which is marked with one of the 4 marks: . ! ? and right (close) quotation mark " (the latter cannot be in character position 1).

**The first sentence is initially defined to be that string which is marked by one of the 4 symbols, whichever comes first. See Fig. 18 for list of valid marks and symbols.

TABLE 2

SENTENCE END MARKER	Rule	1	2	3	4	5	6	7	8	E	*
1.	End of sentence marker for first sentence in Title para. is Period	Y	Y								
2.	End marker is Exclamation Point			Y	Y						
3.	End marker is Question Mark					Y	Y				
4.	End marker is Right Quote Mark							Y	Y		
5.	Valid punc. symbol intervenes (other than one of the 4 end markers)	Y	N	Y	N	Y	N	Y	N		
6.	Go to Table 3	x		x		x		x			
7.	Go to Table 4		x								
8.	Go to Table 6				x		x				
9.	Go to Table 7								x		
10.	Go to Exception Table**									x	

*Means "Else" - the rule to follow when none of the others apply.

**Not yet defined.

TABLE 3

VALID SYMBOL SWITCHES*	Rule	1	2	3	4	5	6	7	8	9	10	E
1.	Next intervening symbol is [	Y										
2.	Next intervening symbol is]		Y									
3.	Next intervening symbol is .			Y								
4.	Next intervening symbol is <				Y							
5.	Next intervening symbol is (					Y						
6.	Next intervening symbol is)						Y					
7.	Next intervening symbol is ;							Y				
8.	Next intervening symbol is ,								Y			
9.	Next intervening symbol is >									Y		
10.	Next intervening symbol is :										Y	
11.	Go to Table 8	x										
12.	Go to Table 9		x				x			x		
13.	Go to Table 10			x								
14.	Go to Table 11				x	x						
15.	Go to Table 12							x				
16.	Go to Table 13								x			
17.	Go to Table 14										x	
18.	Reiterate Table 3 **											x

*For symbols other than the 4 end of sentence markers, except a period. Valid symbols (such as left ") not listed, and non-valid symbols encountered in the string, follow "Else" rule.

**Exit when no further intervening valid symbols exist in the string, or when $b and $c have been set, or any combination of these: $b + no $c found; $c + no $b found. Length limit must always be tested. (The logic for $c is dependent on a further algorithm). In any case, if no further processing can be done for short title, procedure continues to further algorithms.

TABLE 4

PERIOD (FULL STOP)	Rule	1	2	3	4	5	6
1.	Period occurs in char. position 1,2,3, or 4	Y	Y	N	N	N	N
2.	Hit on author statement key words list (test left & right)+	Y	N	Y	Y	N	N
3.	End marker occurs in position compared to string length limit*	-	-	LE	GT	LE	GT
4.	Insert Interim Delimiter after end marker					x	
5.	Do Table 5				x		x
6.	Insert a $c delimiter prior to author statement key word (in "word - 2" position if non-blank in Word - 1 position)	x		x	x		
7.	Go to next end marker		x				
8.	Go to Table 2		x				
9.	Go to [tables for further algorithms, as needed**]	x		x	x	x	x

*Length limit is a parameter for string length of n where n is the maximum number of characters desired in the output for which Short Title is being used, e.g., two lines of text overprinted in a heading on an added entry card.

**Not included in this presentation. A positive determination of the explicit code for the sub-field following short title will often depend on algorithms for the subsequent parts of the title paragraph, including imprint. Moreover, the Interim Delimiter may be deleted altogether if neither Remainder of Title nor Remainder of Title Page Transcription sub-fields are determined to exist in the record at hand. (A Place of Publication sub-field always exists). Finally, a check should always be made at the end for the length limit, which may involve insertion of a $b delimiter on a rather arbitrary basis, in some older records.

+The anomalies that this test must cope with are many: e.g., the period may be embedded, as in 'Ed. by ...'; it may appear to the left, as in 'title. By ...'; and it may be in the title proper, as in 'Philosophies men live by.' or 'Great teachers, portrayed by those who studied...'

TABLE 5

LENGTH LIMIT (CLOSED)	Rule	1	2	3	4	5
1.	Relation of author statement key word to end marker	Hit left	Hit right	Hit Rt.	NoHit	NoHit
2.	Character to which length limit signal points		Blnk	Non-b̸	b̸	Non-b̸
3.	No short title delimiter required	x				
4.	Scan right to left for 1st non-blank character(hopefully an interword b̸)			x		x
5.	Insert $b delimiter in blank position		x	x	x	x
6.	Exit	x	x	x	x	x

TABLE 6

! or ? END MARKER	Rule	1
1.	Insert Interim Delimiter immediately after the ! or ? mark	x
2.	Go to Table 3	x

TABLE 7					
RIGHT QUOTE MARK END MARKER*	Rule	1	2	3	4
1.	Character in next right character position from Right Quote Mark	; or ,	; or,	Blank	Blank
2.	Hit on author statement key words list (test word to right of Right Quote)	Y	N	Y	N
3.	Insert Interim Delimiter to right of punctuation mark in next succeeding character position to right of Right Quote Mark		x		
4.	Insert Interim Delimiter to Right of Right Quote Mark				x
5.	Insert $c delimiter to left of author statement key word (or left bracket if any)	x		x	
6.	Go to Table 3	x	x	x	x

*A left Quote Mark is assumed to have been found in the initial scan of the Title Statement sentence.

TABLE 8						
LEFT BRACKET	Rule	1	2	3	4	5
1.	Digits in character positions 1-4 in Title Statement**	Y	N	N	N	N
2.	Hit on author statement key words	-	Y	Y	N	N
3.	Bracket position: Length limit	-	LE	GT	LE	GT
4.	No $b delimiter needed	x				
5.	Insert Interim Delimiter preceding left bracket				x	
6.	Do Table 5			x		x
7.	Insert $c delimiter preceding left bracket		x	x		
8.	Go to Table 3	x	x	x	x	x

**Also scan to right of Left Bracket for string: "i.e., ..."

TABLE 9								
NEXT SYMBOL =]) or	Rule	1	2	3	4	5	6	7
1.	Interim delimiter already set	Y	N	N	N	N	N	N
2.	Symbol is followed by ";"	-	Y	Y	Y	N	N	-
3.	Symbol is followed by a b̸ then cap letter not in auth. statement key word list	-	Y	N	N	Y	N	-
4.	Hit on author statement key word list (word to right)	-	N	Y	N	N	Y	-
5.	Site counter	-	EQ	EQ	EQ	EQ	EQ	GT
6.	Insert Interim Delimiter after";"		x		x	x		
7.	Insert $c delimiter prior to key word			x			x	
8.	Go to [tables for further algorithms]	x	x	x	x	x	x	
9.	Return to site of corresponding (or <							x
10.	Go to Table 3							x

TABLE 10

PERIOD (NON FULL STOP)	Rule	1	2	3	4	5	6	7	8	9
1.	Interim Delimiter already set	Y	N	N	N	N	N	N	N	N
2.	Period is "inside" to right of string starting with [< (or left "		Y	N	N	N	N	N	N	N
3.	Period foll. by right " then ␢ or ;			Y	Y	N	N	N	N	N
4.	Hit on author statement key words list (test left and right)			Y	N	Y	N	N	N	N
5.	Single preceding character						Y	N	N	N
6.	Single preceding word							Y	N	N
7.	Next word after period is a repeat of first word or root thereof in Title Statement								Y	N
8.	Ellipsis found, or string of interspersed periods									Y
9.	Insert Interim Delimiter after right " or ;				x					
10.	Insert Interim Delimiter after period							x	x	
11.	Insert $c delimiter prior to key word			x		x				
12.	Do Table 15									x
13.	Go to Table 3		x		x		x	x	x	x
14.	Go to [tables for further algorithms]	x		x		x		x	x	x

TABLE 11

	LESS THAN OR LEFT PAREN. Rule	1	2	3
1.	Interim Delimiter already set	Y	N	N
2.	First < or (in Title Statement	-	Y	N
3.	Set site counter	x	x	x
4.	Insert Interim Delimiter prior to < or (		x	
5.	Go to Table 3			x
6.	Go to [tables for further algorithms]	x	x	x

TABLE 12

	SEMI-COLON Rule	1	2	3	4	5	6
1.	Left Bracket pointer "on"	Y					
2.	Left Quote Mark pointer "on"		Y				
3.	< or (site counter non-even			Y			
4.	Interim Delimiter already set				Y		
5.	Semi-colon is first intervening symbol, other than 1-3 above					Y	N
6.	Insert Interim Delimiter after semi-colon					x	
7.	Go to Table 3	x	x	x		x	
8.	Go to [tables for further algorithms]				x		x

TABLE 13

	COMMA Rule	1	2	3	4	5	6	7	8	9	10	E
1.	Left Bracket pointer "on"	Y										
2.	Left Quote Mark pointer "on"		Y									
3.	< or (site counter non-even			Y								
4.	Interim Delimiter already set				Y							
5.	Digit pattern following comma					Y	N	N	N	N		
6.	Comma is 1st interven. symb.						Y	Y	N	N		
7.	Hit on author statement key words						Y	N	Y	N		
	list (test right)											
8.	Hit on multiple title conditions*										Y	
9.	Insert Interim Delimiter after comma									x	x	
10.	Insert $c delimiter after comma						x		x			
	and prior to key word											
11.	Go to Table 3	x	x	x		x		x		x	x	x
12.	Go to [tables for further				x		x		x			x
	algorithms]											

*E.g., combination of ", and □..." where □= Capital letter.

TABLE 14											
COLON Rule	1	2	3	4	5	6	7	8		9	E
1. Left Bracket pointer "on"	Y										
2. Left Quote Mark pointer "on"		Y									
3. ⟨ or (site counter non-even			Y								
4. Interim Delimiter already set				Y							
5. Hit on author statement key					Y	N	N	N		N	
words list											
6. Next is period or ;						Y					
7. Next is ,							Y				
8. Next is [(or ⟨								Y			
9. Next is another colon										Y	
10. Set Interim Delimiter after .;						x					
11. Insert $c delimiter prior to word					x						
12. Set Interim Delimiter after ,							x				
13. Set Int. Delim. prior to [etc.								x			
14 Go to Table 3	x	x	x	x			x	x			x
15 Go to [tables for further					x	x		x			x
algorithms]											
16 Repeat starting at Step 5										x	

TABLE 15		
ELLIPSIS (CLOSED) Rule	1	2
1. 3rd period followed by	Valid symbol	Blank, letter, or
		non-valid symbol
2. Insert Interim Delimiter		x
prior to first period in		
ellipsis		
3. Exit	x	x

[END]

V-3. EXPERIMENTAL ON-LINE MATHEMATICS CITATION DATA BASE*

By
Mary L. Tompkins
Institute of Library Research
University of California
Los Angeles, California

A. INTRODUCTION

A Mathematical Citation Index was started in March 1965 under the auspices of the UCLA Computing Facility with the encouragement and cooperation of the University Research Library, the Engineering and Mathematical Sciences branch library and the Numerical Analysis Research Library. Starting in May 1966, the project was continued by the Institute of Library Research as an example of mechanized services which would be provided by the Center for Information Services in the University Research Library.

It is of particular value because a real data base is accumulating which will be of direct value to mathematicians and other scientists on the campus, as well as contributing to increased expertise in the utilization of large files.

Publication of a series of indexes designed to identify scientific serials has been started. These are generally being identified as MAST: Minimum Abbreviations of Serial Titles. The first volume in this MAST series, on Mathematics, was to appear in 1968, published by Western Periodicals, Inc.

Not only do we foresee the need to publish for wide distribution some products of our work, but we will need to utilize products produced by others. To recreate files already compiled by others is not feasible in terms of time or money. Thus, our planning for an operational citation index at UCLA assumes the purchase and utilization of existing files, or selections from them, such as the I.S.I. Science Citation Index tapes and the Physics citation data being developed by M. M. Kessler at M.I.T. Such acquisitions would then be augmented to satisfy local specialized demands.

B. MAST: MINIMUM ABBREVIATIONS OF SERIAL TITLES

Before the compilation of citations was begun it was evident that some way was needed to identify bibliographic references which were incomplete, ambiguous or otherwise unidentifiable. Hence, the preliminary work on the project concentrated on a circularly shifted index of abbreviations of mathematical journals.

---

*The following material is excerpted from "Experimental On-line Mathematics Citation Data Base," by Mary L. Tompkins. Part 8 of the Final Report on Mechanized Information Services of the University Library, Phase I - Planning. Los Angeles, Institute of Library Research, Dec. 15, 1967. pp. 3-13. Work on this particular phase of the project was supported under NSF Grant GN-503.

What has been widely discussed as a K-W-I-C (Keyword-in-Context) or permuted index, we prefer to call by the more accurately descriptive name Circularly Shifted Index. In this index of abbreviations of serial titles, the abbreviation of every significant word in a title appears alphabetically at a center gutter, with its preceding and following words around it. Each indexed abbreviation is accompanied by an identification number and a short one-line title. The identification number refers to a Reference Section which lists (for the more than 2000 serials) full and complete titles, dates of publication, and histories of title changes.

For any journal with more than one title, corresponding additional abbreviations are entered. Likewise, if a journal is sometimes known by a name not appearing as part of its title, this name would be added. An example of this is "BIT," which is used for the "Nordisk Tidskrift for Informations-Behandling"--the result of a fancy cover design.

Minimum abbreviations (Minabbs) of serial titles and the indexing of them instead of the full titles, is a scheme devised by John W. Tukey at Princeton University and the Bell Telephone Laboratories in connection with his Citation Index for Mathematical Statistics and Probability, and in response to a wide-spread desire for a means of identifying journals cited by unclear and unexpandable abbreviations. The work at UCLA has been a direct outgrowth of the author's work with Professor Tukey.*

Briefly, a minimum abbreviation is formed by using the initial letters of the word up to the second vowel. (For this purpose, contiguous vowels are considered one.) Minabbs are not necessarily the most common abbreviation and they are frequently not unique, but they may be formed from longer abbreviations and from words in any language. As a example: either the title, "Verhandlungen der Naturforschenden Gesellschaft im Basel", or the acceptable abbreviation "Verh. Naturforsch. Gesellsch. Basel" can be shortened to the Minabb VERH NAT GES BAS, which can be located in the Circular Shift Index in any one of four places.

Ambiguity in an abbreviation is useful in this index because it immediately displays where confusion may lie. J MATH PHYS may indicate either the "Journal of Mathematics and Physics (MIT)" or the "Journal of Mathematical Physics (New York)". The reference section indicates that the former dates from 1921; the latter from 1960. A rough volume-year correlation will help determine which serial was intended.

---

*Tompkins, Mary L. and John W. Tukey. "Permuted Circularly-Shifted Indexes: A Mechanically Prepared Aid to Serial Identification". IN: Progress in Information Science and Technology; Proceedings of the American Documentation Institute, 1966 Annual Meeting. Adrianne Press, 1966. pp. 347-355.

Another type of confusion rests in multiple titles. The "Annales Academiae Scientiarum Fennicae Series A" is usually catalogued under its Finnish title "Suomalaisen Tiedeakatemian Toimituksia. Sarja A", but is usually cited under the Latin. The entry under the abbreviation of the first title will lead to the second through the identification number, and thence to the library classification and shelving of the volume.

The results of this phase of the Citation Index project are:

1. A dictionary of acceptable abbreviations for editing input to the Mathematical Citation Index files, for indicating new or omitted titles of mathematical serials and for drawing together in the citation files different versions of a single title.

2. An Institute of Library Research-UCLA Computing Facility report on the computer programs.*

3. Publication of Minimum Abbreviations of Serial Titles: A Circularly Shifted Index to Mathematical Journals.

C. SELECTION OF SOURCE JOURNALS

At the time this project was started "Mathematical Reviews" listed 872 journals which publish research in mathematics. In order to determine how many of these journals constitute the "core" of the literature, several mathematicians at UCLA and Princeton were asked to indicate journals in which most of the significant research appears. Each listed about 100 titles, and a set of 53 journals appeared in all lists. Our "core" consists of these 53 and other multiple but not unanimous choices.

From this an ordering of journals to be processed was established. First, twenty-two current issues of nineteen different journals were scanned and the number of times journals were cited was counted. There were 215 papers which referenced 1294 items distributed among 325 different journals. Those journals most cited were given priority in processing. A slightly changed priority list reflected the rankings after 13,500 citations could be counted. A third count of 22,800 journal citations substantiates these rankings. They are also compatible by and large with those made by Charles H. Brown using 3,168 citations from ten journals published between 1950 and 1952.**

---

*Johnson, G. D., and Tompkins, Mary L. A Circular Shift Index of Abbreviations of Journal Titles: Description of the 7094 Computer Program. ILR-UCLA Computing Facility Report, 1966.

**Brown, Charles Harvey. Scientific Serials. Chicago, Association of College and Research Libraries, 1956.

D. SCOPE OF CITED ITEMS

During the first two phases of this project, 4094 articles were processed. They were drawn from 25 journals published from 1965 to 1967. Five journals were covered from 1965 to mid-1967; nine more were covered for 1965 and 1966; eleven more were covered in 1966 only.

New titles were introduced starting with their current issue, working backward as time allowed. Once undertaken, the journal was completed for the entire volume except when all processing had to stop temporarily in July 1967.

At present 31,424 citations exist on punched cards and on magnetic tape. 22,823 (or 73%) cite journal literature; 5840 (18%) cite books; 3% cite material unpublished at the time of citation; and the remaining 1829 items fall in the miscellaneous category of reports, theses, lecture notes, proceedings of meetings, etc. (These precentages substantiate those compiled on 18,000 citations six months earlier: journals, 71%; books, 18%; and, unpublished, 3-1/2%.)

E. PROCESSING CITATIONS

Material for the Citation Index is keypunched on cards which are later transferred to magnetic tapes, disks or data cell for computer manipulation and storage.

Current issues of scientific journals have been the sole source of citations and 73% of the input citations have referred to scientific journals. Therefore, the card format and the experiments with on-line input have been geared to journal identification. It is possible, however, that bibliographies from other sources, particularly from proceedings of meetings or Festschrifts, may be desirable for inclusion. The format for a citing journal is readily adaptable to a book by merely assigning an arbitrary seven character designation to the item to correspond with the journal serial and volume numbers. This number with that of the first page serves to bind related citations to the source item.*

The author and title of the citing paper and of each referenced work (excepting only personal communications) are recorded. An attempt is made to preserve all bibliographic information given by the author. That which will not fit conveniently on the rigidly formatted citation or C-card is expanded in full text on supplementary or S-cards. Extra long names, or multiple authors which exceed the 15 character cited-author field are recorded on name or N-cards.

---

*NOTE: The data record formats used in the early stages of the UCLA Citation Indexing Project were developed in advance of the Library of Congress MARC format. It is expected that analysis of the feasibility of adapting a version of the MARC II format to the journal article file format will be undertaken during a later phase of the File Organization Project so that a uniform record structure concept can be tested in an on-line environment, and for purposes of making the data base convertible to MARC II for transmission to other agencies.

Fig. 1 illustrates the cards punched for citations from a single paper.

F. USE OF DISPLAY STATIONS FOR ON-LINE EDITING

Programming the IBM 2075 for the Mathematical Citation Index has been designed to exploit the display potential of cathode ray tube terminals. These instruments provide on-line visual access, query, and input to computer stored data. Man-machine communication with them requires a minimum of special operator skills. They can be used geographically distant from the computer.

The program we call CITATION is an entry into the UCLA time sharing system. It is designed to facilitate input of new data into the Citation Index file or correction of existing records. It edits input records, refusing to store illegal numerical data. An illegal entry may be one that is too long, or too small, or one that contains a blank or an embedded character. A message indicating the error will appear on the screen until correction is made.

Separate formats accommodate information about cited and citing items. These have been based on the punched card format for easier conversion. Since journal articles predominate, attention has been paid to their peculiar bibliographic conventions.

The program commences by clearing the screen (buffer) and displaying MAIN TITLE:

```
                CITATION INDEX

        TYPE A FOR CITING AUTHOR FORMAT
             T FOR TITLE OF CITING PAPER
             C FOR CITATION FORMAT
             R FOR CITED TITLE
             I TO INITIALIZE
             D TO DISPLAY
             S TO STORE
```

When the letter I is typed the screen displays:

```
                JOURNAL FORMAT

                JOURNAL NUMBER
                VOLUME NUMBER
                YEAR
```

FIGURE 1:

CARDS PUNCHED FOR ONE PAPER PUBLISHED IN VOL. 66 OF THE COMMUNICATIONS IN PURE AND APPLIED MATHEMATICS

For the Citing Paper

Field
1. Type Card (A1 = Author; T1 = title)
2. Type Number
3. Year
4. Journal
5. Volume
6. First Page
7. Last Page
8. Author name or title statement

For its Cited Items

Field
1. Type of Card
2. No. of Citation
3. Citing Journal
4. Citing Volume
5. Citing Page
6. Author
7. Designation of Multiple Author
8. Designation of Non-journal
9. Abbreviation of Location (title of source of cited item)
10. Volume
11. Issue
12. Year
13. First Page
14. Last Page

5

The volume, year and assigned identification number of the journal containing the citations to be recorded are entered in the INITIALIZE routine. These numbers provide the link between cited and citing papers and automatically become a part of every succeeding record until the program is reinitialized.

Typing letter A presents the AUTHOR display:

CITING AUTHOR INPUT

FIRST PAGE NUMBER
LAST PAGE NUMBER
YEAR

Entered page numbers are checked to be sure the last is at least as large as the first. Multiple authors may be listed.

The TITLE display is simply:

TITLE OF CITING PAPER

and allows for 440 characters of free text.

The citations are now entered on the following format.

CITATION INPUT

VOLUME
YEAR
ISSUE, EDITION
FIRST PAGE CITED
LAST PAGE CITED
JOURNAL ABBV
AUTHORS

The CITATION FORMAT is displayed automatically with serially increasing numbers for each entry until A is called, for the next citing author input. Before the next series of citations can be entered, however, a subtitle asks the operator,

ARE YOU STILL IN THE SAME
VOLUME AND JOURNAL? IF YOU ARE
NOT, PLEASE RE-INITIALIZE.

The editing routines of this program are being constantly expanded. The next powerful addition to the system will be a check of journal abbreviations to insure that all citations to a given paper are being lumped together regardless of the vagaries of referencing by the citing authors.

APPENDIX VI

SAMPLE SIZE DETERMINATION FOR DATA CONVERSION QUALITY CONTROL

By

Jorge Rodriquez
Institute of Library Research
University of California
Berkeley, California

SAMPLE SIZE DETERMINATION FOR DATA CONVERSION QUALITY CONTROL

By

Jorge Rodriguez
Institute of Library Research
University of California
Berkeley, California

A. INTRODUCTION

This analysis aims to solve a specific problem of the Institute of Library Research, summarized as follows: it is desired to investigate the possibility of using a statistical control method to determine the accuracy of the data conversion process and to estimate the frequency of errors associated with a common source.

The constraining factor involved in the control process is the cost of sampling as opposed to the cost fluctuation of the conversion process as a function of the level of accuracy.

B. DISCUSSION

It has been suggested that sampling inspection, and more specifically, sequential sampling inspection, be studied as a possible control method. It will later be explained that the nature of the problem suggests other means of control.

In sampling inspection, whether it is single, double, or sequential, we normally start with two fixed points. Assuming that the parameters in question lie on the mean of a distribution (as in the present case) these two points are: a previously estimated mean value, which is assumed to remain fairly constant and considered acceptable; and the worst value of the mean that will be considered acceptable. This introduces the notion of acceptance and rejection of the lot in question.

The test consists of obtaining a sample from which the mean of the population is estimated using a proper estimator. Then, using the lot size and sample size, obtain some measure of the quality of the decision. For this purpose, two types of errors are defined and their probabilities estimated: namely, the probability of error of type I, which is the probability of rejecting a good lot; and the probability of error of type II, which represents the probability of accepting a bad lot.

The situation in the data conversion is somewhat different. In the present problem it is not assumed that the mean of the distribution remains fairly constant, nor that there is a region of acceptance opposed to one of rejection. The problem here is more related to point estimation, to the actual estimation of the expected number of inaccurate records, and to obtaining some measure of the estimate in the form of a probability that deviates by an amount less than Δ .

C. PROPOSAL

Consequently, it seems appropriate to use a simple point estimation procedure. In this procedure, the size of the sample will be determined by economic factors. Then the weight of the estimate will be measured by fixing a deviation Δ , and evaluating the probability of the estimate being within a plus or minus Δ from the exact value.

The following is an analysis and detailed description of the steps of the proposed method.

1. Selection of Lot Size N. The lot size will be determined by the frequency at which the control is performed, and/or by the total number of records to be processed. This decision is not critical as long as it is over a reasonably large number of sampled records (not less than 1000). In general though, it is desirable to use as large a lot size as possible.

2. Selection of the Sample Size. The sample size will mainly be determined by economic considerations, and for this purpose a cost function is defined which includes all the variable costs of interest:

$$C_t = f_1(m) - f_2(\hat{M}) + f_3(n) \qquad (1)$$

where,

$f_1(M)$ = cost function representing the cost of correcting M wrong records.

M = exact number of wrong records due to the conversion process.

$\hat{M}$ = estimate of M.

$f_2(\hat{M})$ = cost function representing the discount when the number of wrong records due to the conversion process is $\hat{M}$.

n = sample size

$f_3(n)$ = cost function representing the cost of obtaining a minimum sample of size n.

It is reasonable to adopt a minimum sample size. In other words, the estimate will not be accepted if it is obtained using a sample smaller than the lower limit. The first step, then, is to obtain a sample whose size is equal to the lower limit and to estimate the number of wrong records by using a simple average as the estimator, or

$$\hat{M} = (\frac{m}{n})\,N \qquad (2)$$

where m = number of defective records in the sample (records containing at least one error).

The next step is to investigate the possibility of increasing the sample size by comparing the cost of obtaining the previous sample size with the expected saving, which can be expressed by

$$S = f_2(\hat{M}) - f_1(\hat{M}) \tag{3}$$

If this value is less than $f_3(n)$, then a larger sample is not economically justified, and we proceed to para. 3. If S is larger than $f_3(n)$, then the sample size can be increased to some value n' such that:

$$f_3(n') = S \tag{4}$$

We can obtain a new estimate of the number of defective records (M') by using equation (2) again

$$M' = \left(\frac{m'}{n'}\right) N \tag{5}$$

where m' is the total number of defective records in the sample of size n'.

By repeating the process of comparing sampling costs with expected saving we can determine whether it is justified to increase the sample size.

It is also reasonable to set an upper limit on the sample size. In other words, the sample will not be increased beyond that limit even if it is economically feasible.

3. Measure of the Estimate $\hat{M}$. Once we have obtained the sample size n' and the number of defective records in the sample m, we can obtain a statistical measure of the estimate by using a convenient sampling table.

The table referenced (1) gives a relationship of five parameters: N, n', m', confidence level, and the precision. Any four of them will determine the other.

In our case we already have N, m' and n'. And from the structure of the table, it is easier to decide on the confidence level desired and then obtain from the table the precision.

To clarify the meaning of each of these terms, the following equation is presented, showing the relationship among them:

$$P\left\{|M - \hat{M}| \leq (\text{Precision})\, \hat{M} \,\middle|\, \frac{\hat{M}}{N} = \frac{m}{n} \text{ and the population is } N\right\} = \text{confidence level} \tag{6}$$

D. ILLUSTRATIVE EXAMPLE

1. Given: 200,000 records to be processed in 50 weeks.
Cost of correcting one wrong record = \$0.10
Discount per wrong record = \$0.25
Cost of sampling = \$0.05/record.
It is desirable to obtain a control output every week.

2. Solution.
Select the lot size N, which is given by:

$$\frac{200{,}000 \text{ records}}{50 \text{ weeks}} = 4{,}000 \text{ records/week}$$

Select lower and upper bounds for sample size, and set these at 5% and 20% of the lot size, respectively.

Now, to perform a control for one week we proceed as follows: By using a table of random numbers we select a sample size of $n = 200$. Assume that we found 9 defective records in the sample, then $m = 9$. We obtain the first estimate by using equation (2) where,

$$\hat{M} = \left(\frac{m}{n}\right) N = \left(\frac{9}{200}\right) 4000 = 180.$$

By using equation (3) we evaluate the expected savings:

$$S = 180\ (25) - 180\ (10) = \$27.00$$

Evaluate $f_3(n) = 200\ (5) = \$10.00$

Comparing S and $f_3(n)$ we see that it is justifiable to increase the sample size in order to get a better estimate. By using equation (4) we obtain a new sample size.

$n'(5) = 2700¢$, or $n' = 540$, which is only about 15% of the lot size. We obtained 340 more records. Suppose that we obtain 15 more defective records. Then $m' = 15 + 9 = 24$, and our new estimate from equation (5) will be $M' = 178$.

The new expected savings are then $S' = \$26.70$, which is less than $f_3(540) = 2700$, and therefore we do not increase the sample size.

Now, suppose we want to obtain the precision of the estimate with a confidence level of 95%.

To use the table, first we calculate p, the proportion of defective records in the sample

$$P = \frac{m'}{n'} = \frac{24}{540}\ (100) = 4.4\%$$

Rounding this to the larger integer, to be on the conservative side, we have $p = 5$.

Using Table I, page 23 of the referenced source we find that if n were 409 the precision would be 2%. Consequently, we conclude that $100 < M < 260$, with a confidence level better than 95%.

$$[\, M - .02M \leq M \leq M + .02M \,]$$

If it is desired to estimate the errors attributable to each type of source, we need only to use formula (2) with the corresponding m's, and by using the tables obtain a measure of the estimates.

REFERENCE

1. Brown, R. Gene and Lawrence L. Vance. Sampling Tables for Estimating Error Rates or Other Proportions. Berkeley, Public Accounting Research Project, Institute of Business and Economic Research, University of California, c1961. p. 23.

APPENDIX VII

THE ORGANIZATION, MAINTENANCE AND SEARCH OF MACHINE FILES

By

Ralph M. Shoffner
Institute of Library Research
University of California
Berkeley, California

(Published in the Annual Review of Information Science and Technology, v.3, edited by Carlos A. Cuadra. Chicago, Encyclopaedia Briticanna, Inc., 1968. pp. 137-167)

UNIVERSITY OF CALIFORNIA
INSTITUTE OF LIBRARY RESEARCH
Berkeley, California 94720

Reprinted from The Annual Review of Information Science and Technology, Vol. 3, 1968, edited by Carlos A. Cuadra, by permission of Encyclopaedia Britannica, Inc. Copyrighted 1968.

Permission to reproduce this copyrighted material has been granted by Encyclopaedia Britannica, Inc., to the Educational Resources Information Center (ERIC) and to the organization operating under contract with the Office of Education to reproduce ERIC documents. Reproduction by users of any copyrighted material contained in documents disseminated through the ERIC system requires permission of the copyright owner.

5

The Organization, Maintenance and Search of Machine Files[1]

RALPH M. SHOFFNER[2]
University of California
Berkeley, California

Reprinted From
ANNUAL REVIEW OF INFORMATION SCIENCE
& TECHNOLOGY VOL. 3
Carlos Cuadra - Editor
© 1968 Encyclopaedia Britannica Inc.

INTRODUCTION

File organization and search is concerned with the structure and operation of computer systems that store and retrieve large amounts of information. The following are four key questions about file systems that, ideally, a review article of the current literature should answer: How well do such systems operate? What are the structures of file systems? What is the nature of the system operation? How does one design an appropriate system?

Unfortunately, there are no simple answers to these questions—first, because the literature contains only partial answers, and second, because of the great diversity among file systems and file system languages. These two reasons give rise to a host of other reasons:

1. With many new people working in the field, undefined vocabulary increases more rapidly than strict definition reduces it.
2. File systems have to be designed to resolve contradictory goals, such as rapid access, high capacity, low user training, and high reliability.
3. The machine system does more of the processing to organize and group related information.
4. There is an increase in different kinds of information to be processed.

---

[1] This review was supported in part by the U.S. Office of Education Grant No. OEG-1-7-071083-5068.
[2] I am indebted to Luke T. Howe for his extensive help in preparing this review.

5. Desired modes of access to information are increasing.
6. Machine processes are being developed to reduce the need for experienced specialists to act as broker between the users and the systems.
7. Even with a determined file structure, the performance of large-scale, multiple-access file systems can be significantly affected by the search procedures used.
8. The expanding computer field means there are more computer systems and programming languages with which to implement file systems.
9. Many special-purpose systems are being developed and their reports are being published.
10. In spite of all these trends, that Holy Grail—the all-purpose data management system—is being sought by an ever-increasing number of people.

One evidence of the increasing interest in file structures is the establishment by the Association for Computing Machinery of a one-day professional development seminar in "File Structures for On-line Systems." As with the other ACM seminars, it was scheduled and presented in several cities throughout the United States. The material was well organized and the meetings were well attended. No doubt the specific material will change, but seminars should certainly be continued. Another important event in 1967 was the publication of Meadow's excellent book (60) devoted to file organization. It will serve as a text around which file systems courses can be organized. These courses will also be able to make good use of several other recent books that devote sections to file organization and search (37, 72).

This review is divided into three sections: General Issues, Aspects of File Systems, and File Systems. The division between the last two areas is somewhat artificial, both in the sense that some of the literature must be discussed in more than one place and in the sense that one aspect of a report is emphasized to the exclusion of other aspects where the chosen aspect seemed most important. However, rather than attempting to give a balanced review of individual papers, we have tried to report significant changes occurring over the entire literature of file organization.

GENERAL ISSUES

Evaluation of Systems

Although a separate chapter of this volume (Chapter 3) is devoted to evaluation, a portion of that literature needs to be considered here because of its potential importance in system design. The growth of machine files has increased the capability for making partial searches, and it is important to be able to determine the extent to which file structure and search techniques influence recall, precision, and other measures of system performance.

Pollock (68) provides a brief but excellent review of some of the measures which have been used for the evaluation of systems. He then defines another, a normalized "sliding" ratio measure. Instead of having two classes, his measure has multiple classes to which each retrieved document is assigned according to its relevance to the query. This approach should prove useful since it avoids the difficulty of having to judge a document as completely relevant or irrelevant.

The primary measures for the evaluation of document retrieval systems continue to be those of recall and precision. Swets (79) provides an interesting representation of these measures, in which, by a change of variables, he plots an operating characteristic curve of the type used in statistics. His objective is to simplify the task of evaluation by replacing the two measures with a single measure of performance. This, he argues, can be done by treating the slopes of the operating characteristic curves as equal. If they are equal, they can of course be ignored. However, his data do not appear to justify this treatment. That is, a different selection method will provide the best performance, depending upon the ratio of recall to precision one is willing to accept.

As an alternative to the use of recall and precision for the measurement of information transfer, Hayes (37) has suggested an abstract measure that takes into account the amount of information in the search specification, the total size of the file from which the result is selected, and the size of the resulting retrieval.

While such an approach may prove useful, the measure developed still will not incorporate the issues of system and user time and cost. So long as this is the case, evaluation will be largely irrelevant to the design of file systems. Hayes recognizes this in his introduction:

> The third aspect—organization—arises because as the file gets large enough, it is impossible, at least uneconomical, to scan every item in the file to judge its relevancy. It is therefore necessary to structure the file to provide indexing mechanisms, and to provide intermediate measures of degree of match which are less sophisticated than the ultimate measure of relevancy. It is this third aspect which really constitutes the technical problem in information retrieval system design, since it is here that the size of the file, the requisite response time, the degree of selectivity, and the accuracy of the response all interact. (p. 265)

When measures incorporating all of these aspects are developed, they will provide a quantitative basis for file systems design. Until that time, design will continue as a highly subjective process.

Measurement of Association

The measurement of association is central to the organization and search of machine files. Such measurement is needed in order to associate file records with each other and to associate them with a search request. Usually this measurement is of the match-mismatch type. That is, either a given file record matches the search specification completely or it is considered a mismatch. In this case, no estimate of the relative mismatch of the file records is made.

While most operating file systems have been based on exact match, considerable experimental work with statistical measures of association has been performed. As a result of this work, associative measures have reached the stage where they will be used in operating file systems. Jones & Curtice (41) provide a comparison of term association measures. The authors use a framework that encompasses most associative measures. They define a weighting in which the measure of association of two index terms is equal to the frequency of the joint occurrence of the terms divided by the nth power of the frequency of one of the terms. By varying the value of n, the behavior of the various associative measures can be approximated. The authors provide an illustrative example, using a term taken from the NASA collection, which contains 100,000 documents and 18,000 index terms. In addition to providing useful approximations, the authors also show that a specific value of n in the approximation formula implies a weighting of the relative importance of recall versus precision in a search operation.

Salton's work (71) gives experimental evidence of the usefulness of associative techniques.

> Specifically, the procedures based on synonym recognition, weighted content identifiers, cosine correlation to match documents and search requests, and document abstract processing are *always* more effective than methods using simple word stem matches (without synonym detection), nonweighted terms, correlation terms based only on the number of matching terms, and analysis procedures which consider only the titles of the documents being examined. (p. I–4)

The importance of these procedures for operational systems appears to be in the order in which they are mentioned. In general, the importance of synonyms is already recognized. To a limited extent, weighted index terms are already in use. The next important step will be the incorporation of term pairs with associated frequency information so that requests tailored for recall and precision can be satisfied without significant increases in search cost.

Research on associative measurement is continuing, with two major areas of investigation. One is the machine analysis of text to obtain the appropriate index terms. (This work is covered in Chapter 6, on automated language processing.) The other is iterative search, in which the retrieval results and the associative measure are used to modify either the search request or the indexing provided in the file. In addition to Salton's group, groups reporting on associative adjustment systems include Jones, et al. (42), Bryant, et al. (16, 73) and Lehigh University (2). The work reported by Salton's group is of the greatest interest in that experimental results are provided for techniques utilizing user feedback for the modification of search.

ASPECTS OF FILE SYSTEMS

Logical Record Encoding

Record encoding has two separate components. The first component is the specification of the logical content of the record. The literature concerned with the determination of logical content is found in the preceding chapter on document description and representation. However, a few documents are discussed in this section because they have as a common characteristic emphasis upon the development of a hierarchical classification system for encoding the record information. The second component of record encoding is that of the representation of the record as it is held in computer storage. This physical encoding is discussed in the next section.

Two papers report on the experimental application of the Universal Decimal Classification (UDC) to machine search systems. Freeman & Atherton (32) report on its application to 250 documents in the field of oceanography, and Caless & Kirk (17) report on its application to an unknown number of seismological documents at the VELA Seismic Information Analysis Center (VESIAC), University of Michigan. Both papers conclude that the UDC can be used for encoding and searching documents. However, Freeman & Atherton point out that there is an unanswered question of whether it should be used in preference to an indexing language designed specifically for machine processing. Both papers give an indication of some of the difficulties in adapting UDC for machine search. Caless & Kirk indicate that extensive skills and preparation were needed. The skills were in the subject area, in the application of UDC, and in library classification. Their system is highly dependent upon a skilled staff performing the encoding both of the documents and of the search requests.

This dependence upon skilled staff is not unique to the UDC system, but, rather, is inherent in any highly structured representation of document content. The proper conclusion from this is not necessarily that such systems should be abandoned, but, rather, that when they are used, specialized information centers such as VESIAC should be set up. In this way, the record encoding can be performed, in all of its complexity, by information specialists and then can be distributed and used wherever there is need for that literature. This, of course, is the direction in which machine information systems have been proceeding during the last decade.

Just as many problems are encountered in the application of UDC, many problems have been encountered in the use of classification structures for chemical information systems. In general, the literature of the current year represents a continuation of work reported by Tate (80). Bowman (13) reports on the use of the Wiswesser line notation for 100,000 chemical compounds at Dow Chemical. He gives costs for their method of encoding the chemical structure information. Starting with a legible structure diagram and a molecular formula, only $176 was required to code and check the notations and formulas for 1,000 compounds. He does not indicate the length of training time for the encoders, nor does he indicate how familiar they were with the process prior to its operation.

Lefkovitz (48, 49, 50) compares the two main approaches to the encoding of compounds, the connection table and the line notation methods, with respect to the requirements of a system handling approximately 3,000,000. In addition to the two basic approaches, he discusses a derivative system called the Mechanical-Chemical Code (MCC). Although not as complicated as the others, this code could serve as a rough screen in an on-line retrieval system.

Two reports of the continuing work at Canadian Industries, Ltd., appeared during the year (39, 59, 82). The objective of this work has been the automatic generation of a connection table from the Wiswesser notation. This table is to be used in place of the original notation for internal storage, search, and display purposes.

A study at Chemical Abstracts Service is concerned with automatic translation from one representation to another. This one is concerned with the conversion of traditional systematic nomenclature. Vander Stouw, et al. (86) provide a preliminary discussion of this study, which should produce some generally useful results.

Bobka & Subramaniam (10) report on the development of a chemical coding system, called Medical Coding System (MCS). The system, which was developed in the Comparative Systems Laboratory of Case Western Reserve's Center for Documentation and Communication Research, is based upon a classification of chemicals by the type of units or groups present. It does not record the sequence in which the functional units appear; rather, each group is coded individually without reference to other parts of the molecule. No comparison with other notation schemes is made, and it is not clear why a different system was felt to be necessary for encoding the compounds. Presumably some of the experimental and operating systems that are capable of incorporating more than one type of notation system will provide some of the comparative information necessary to clarify the advantages and disadvantages of the various notational schemes (38, 87).

Physical Record Encoding

Character Encoding. Physical encoding is concerned with the binary representation used to encode the data within the computer system. Most systems have used encoding schemes in which characters are the units that are independently represented. For alphanumeric data the most common coding systems are the well-known Binary Coded Decimal (BCD), and the American Standard Code for Information Interchange (ASCII) six-bit and eight-bit codes. Morenoff & McLean (64) recommend an Information Processing Code (IPC), an eight-bit code constructed so that subsets of seven, six, five, and four bits can be derived from it. While defined as an eight-bit code, the eighth bit is not used, and thus it is the seven-bit subset that is presented in the paper. The intention of the authors is not to replace the function of ASCII as an information interchange standard; rather, it is to use IPC for internal information storage, particularly in on-line systems. As the authors indicate, the internal information processing and system-user interface requirements of on-line systems are of sufficient importance to warrant consideration of specially designed codes to satisfy them. As the number and size of on-line file systems increase, further attention to the internal codes can be anticipated.

Word Encoding. Whereas a character-by-character representation of nonnumeric information is likely to be the most desirable for manipulation within the central processing unit and for output over the various peripheral units, the increasing use of mass storage will foster the development of coding schemes that require less storage than character-by-character encoding. As is generally known, the encoding of natural language is not compact because our words use only a small proportion of the very large number of possible combinations of the 26 letters of the alphabet. One approach to obtain a more compact representation has been to develop systems that encode word-by-word rather than character-by-character. These are often referred to as code-compression systems.

Although information theory provides a sound basis for the development of efficient coding structures, the file systems literature reveals little use of this theory. Bemer (6) devotes his paper to the use of compressed codes to reduce the cost of long-line communication of natural language. He shows that if the codes could be generated properly, variable-length coding would be the most efficient because it can take advantage of the frequency of use of words in the natural language. By this approach, the codes are compressed to 35% of the space required for a character-by-character representation. Although he does not discuss the compression technique, he apparently assumes that all programs and tables fit into the computer's main memory.

Bemer estimates total coding and decoding time at 250 microseconds per word, on an IBM 7090. Using a cost of $800 per hour, he computes the total conversion cost to be $0.0056 per word. While the estimates were made for a different purpose, they can be used to indicate the potential of code compression for file system storage. In the interval since this work was done, the cost of equivalent processing has been reduced by approximately 90%. Assuming that encoding and decoding times are roughly equivalent, encoding should now cost approximately $0.0003 per word.

If the annual cost of the mass storage is $6 per thousand words, then a code-compression strategy that reduces a space to less than half of the original would provide a storage cost reduction of more than $3 per thousand words per year. Obviously, retranslation and other requirements must be considered in order to determine the effectiveness of code compression for file systems. Even so, it is clear that systems with mass random-access storage could benefit from attention to code compression.

In some cases, non-unique codes may be useful in file systems. In this area, Bourne & Ford (12) report experimental work showing the degree of non-uniqueness as a function of the length of code generated. The analyses were performed on files of 2,082 index terms and of 8,207 student names. The transformations used were variant methods of selecting letters to be dropped from the source word. The letters were chosen either on the basis of position or on the basis of frequency of occurrence. They show that savings in space—in an index, for example—can be achieved without a great loss of uniqueness. Marron, et al. (58, 28) describe COPAK, a three-component code-compression system. One component, NUPAK, converts fixed-point data to a compact form. Another, ANPAK, compresses alphanumeric data, either the output from the first component or any other fixed string. The third component, IOPAK, provides a final compression of the strings before output to a storage medium. At present, NUPAK and ANPAK have been implemented.

ANPAK is the component of greatest interest for file systems. ANPAK operates repetitively on the string of data and removes repeating elements from the string. The generated code contains all information necessary for decoding without loss of original information. The authors give an example of the use of ANPAK on text material for which character-by-character encoding required 15,180 bits. This was encoded in 9,291 bits, representing a compression to 61% of the original size. No indication of the processing requirements is given. However, the authors assert that such compression is economically feasible, since read-in time for compressed information plus time for decompression is significantly less than read-in time for the original information.

Field Encoding. Beyond the representation of the information on a character or word basis, the next problem is that of mapping the logical fields of the records into internal storage. Benner (7) provides a design technique for mapping logical records into physical records of fixed size. By this method, a common program can perform all direct-access file handling regardless of the specific content or size of the logical records. This approach encounters the problem of determining the size of a physical record and the fields of a logical record that it should contain. Benner's technique deals quantitatively with this issue in terms of the lengths of the control versus the data portion of the fields, the activity of the fields, and the distribution of field lengths. These variables are used to balance storage utilization against access. From this balancing, the design of the record is established. The author describes the results achieved using this technique in the design of a business information system and estimates storage utilization to be 65%. This is the expectation that a character of storage will be occupied by a character of useful information from the data base (excluding control information). The length chosen for the physical record was 796 bytes. The ratio of file processing time to application program execution time is 1:1.2. The system operates with a mean inquiry response time of 10 seconds.

Graphic Data Encoding. The rapid development of peripheral equipment providing input and output of data in graphical form has stimulated interest in the development of data structures for the storage of graphical information. As Van Dam & Evans (85) indicate, data structures for storing line drawings have been at one of two extremes. They have been either descriptions suitable for direct output to a specific display device, or hierarchical, interconnected list structures. The authors provide another list structure, but one with less structure provided within the data. They achieve this by defining a set of primitives that operate on the data to develop the more complex structure. These primitives constitute the Pictorial ENCodIng Language (PENCIL), major portions of which have been implemented on an IBM 7040. In PENCIL, the unit of data is a picture composed of points, lines, and other pictures. A picture is represented by a Control Item and associated Line Items, Text Items, and Information Items. Through the use of PENCIL primitives, one can define or establish new data in the form of points, lines, etc. One can also manipulate the data to move lines and points on the screen and can perform affine transformations. Control can be used to clear working storage, assign names to pictures, retrieve pictures from storage, display them on an output device, delete components of the picture, etc.

The PENCIL approach is particularly interesting because it is integrated within the more general MULTILANG system at the University of Pennsylvania. Though MULTILANG is experimental, such mixed data systems soon will be common. Therefore, file system designers must begin to give more attention to the storage and retrieval of graphical information as well as of alphanumeric information.

The Logical Grouping of Records

To facilitate retrieval, it is desirable to group records that have the same or related content. Where the content of a record is identified by a set of assigned index terms, a common method of grouping the records is to list them under each assigned index term. For systems of this type, Zunde, et al. (94) define the distribution of index terms assigned to documents to maximize information transmission. On the basis of information theory argument, they obtain an equation for the desirable number of records to which a term is assigned as a function of the average number of records assigned per term for the entire file. This distribution is then compared against the distribution of terms in two systems. Although one would expect a uniform assignment of terms to be the most efficient, they do not explain the reason this does not occur.

Long, et al. (53) have analyzed the rate at which the number of words in a dictionary increases with the amounts of text analyzed. Their analysis is part of an attempt to develop a set of keywords for the indexing of radiological patient records. They have encountered difficulty in the use of word-rank order to establish access in that, beyond the first several hundred words, about 70% of the remaining words occur with a frequency of less than one in 10,000. They anticipate continuing the study to determine whether filter techniques can be developed to give (approximately) 2,500 words with which to characterize the file.

In a variation of analyzing text in order to determine appropriate terms, Armitage & Lynch (4) present the development of articulated subject indexes by manipulating phrases. They show that for subject entries from *Chemical Abstracts*, at least half of the articulated terms (i.e., a main term with a subhead) can be generated from more general phrases by the use of their rules. They have also applied the technique to 479 abstracts from

Documentation Abstracts, and they conclude that an index can be developed from the titles of these abstracts.

Taken together, these papers show that a limited analysis of text can be used to determine the logical grouping of similar records in diverse subject areas.

Beyond the grouping of records that have individual index terms in common, it is possible to group records based upon a determination of general similarity. One of the continuing studies is carried on at the Cambridge Language Research Unit. Sparck Jones & Jackson (75, 76) report on the current status of the Unit's clump-finding investigations. Clustering is another term that has been used to describe grouping procedures. Ide, et al., and Grauer & Messier (in Salton, 71, same as 26) report on the current state of the clustering programs under investigation at Cornell. These programs are based upon the clustering algorithm developed by J. J. Rocchio. This clustering algorithm is an alternative to the Cambridge Language Research Unit's clumping approach. It is being tested on the same data base used in the clumping investigation: Cleverdon's collection of 1,400 documents from the Cranfield project. From this, a comparison of the other two approaches should be possible.

A different approach to the grouping of records is that provided by a document's citations. These citations provide linkages between records that can be used to infer similar content. Part of the current work is concerned with the processing of common citations to infer similar content. Thus, the objective is the same as that of clustering, but a different kind of linkage information is used. Chien & Preparata (22, 69) use graph theory to define a procedure for grouping documents by their "distance." It is not known whether any test has been made of the algorithm developed. However, it would be most useful to apply the Chien-Preparata algorithm to the index terms of the Cleverdon documents in order to compare them with the methods of clustering and of clumping.

The citation approach is based upon the processing of "citing/cited by" relations that exist between records in a file. Levien & Maron (52) have generalized this approach in their discussion of a system for inference processing. In this system, relations that exist between records in the file are processed to derive information not explicitly contained in the file. Although a wide range of problems remain to be studied, their approach is important because it reduces the necessity to establish explicitly all of the record groupings in the file structure.

Two papers are concerned with the grouping of records representing chemical compounds. Uchida, et al. (84) are concerned with the evaluation of fragmentation codes, linear ciphers and atom-by-atom topological codes. Particularly, they have been concerned with a system capable of retrieving material having common substructures. They discuss the results of 28 searches in a file containing 841 alkaloids and their derivatives. Armitage, et al. (3) report on the development of algorithms to detect similarities among chemical compounds. Their purpose is to determine the largest connected set of atoms and bonds common to any pair of chemical structures. As investigations such as these proceed, the concepts of similarity of chemical compounds should be developed enough to allow grouping of compounds with similar substructures. This grouping will reduce considerably the required search effort.

File Structure

Given that the records and the logical grouping of these records have been defined, the task remaining is to obtain a machine file to map this logical structure into the physical structure of the computer system. In contrast to records that are characteristically free in form with many partial connections, the physical records of a computer system are normally fixed and rigid. Thus, for example, the physical records of a computer system's mass storage have a given number of bits for each physical record. In this situation, it is most unlikely that the logical records of the external world will match the physical records of the computer system. Thus, computer programs must be provided to translate the records from one form to the other. The term "file structure" refers to the methods by which these programs retain the logical records and their groupings within the rigid system structure.

Two major concerns are reflected in the current literature. One is the development of more general, and more complex, file structures. The other is the analysis of the effect of structure on access time.

Most of the current structure development involves variants of a structure in which each logical record contains the information needed to generate the address of the next logical record linked to it. Such structures have been referred to as structured files, threaded-list files, multi-list files, etc. Gray (35) provides a brief review of many of the list-structure approaches that have been used in computer-aided design. They include the sketch-pad ring structures, the coral-ring structures, and numerous others. This is a most useful paper because it shows the many similarities between these various approaches.

A number of specific list-linking techniques have been reported on during the year. Gabrini (33) reports on an application of Multi-list, a linked-list file system developed at the University of Pennsylvania, to the Project TIP file of *Physical Review* articles. Ross (70) describes the AED free storage package, which is part of the AED-1 compiler system. The package is general in that the blocks may be of any size and the programs provide several different strategies for maintaining storage accounting. Storage accounting and the reassembly of released storage into usable units is generally a problem in such systems. Haddon & Waite (36) discuss their procedure for reallocating variable-length records in order to free space that was previously unallocatable because of its distribution between these records. They describe the general procedure and give times for its use on an English Electric KDF9. As described, the procedure is applied to the fast memory of the central processing unit. Its extension to mass storage is not discussed.

As list-processing languages make increasing use of mass storage, they become more like general processors using list-linked structures. Two papers related to list processing are concerned with the allocation of data to mass storage or to fast memory. Cohen (24) discusses the results of a program on an IBM 7044 with disc storage that allocates the records on the basis of their frequency of utilization. Bobrow & Murphy (11) discuss a similar application that utilizes a DEC PDP-1 with a drum memory. Both conclude that frequency can be effectively used to allocate space.

Just as both of these approaches were heavily influenced by the MULTICS system, so too was the system described by Barron, et al. (5). They summarize the file handling facility that is provided on the Titan computer at the University Mathematical Laboratory, Cambridge. In this system, files are maintained over indefinite periods on a three-level system having core, disc, and magnetic tape. Access is provided on the basis of the name of a desired logical block of information—i.e., a group of records. The system maintains necessary indexes to provide the physical records associated with the name, and to reallocate the physical records among the peripheral storage units.

Morenoff & McLean (63) discuss multi-level storage organization in a more abstract fashion. They suggest the definition of levels of the file in terms of the accessibility characteristics of the organization. On this basis, they obtain something around 30 to 40 levels, with access times from nanoseconds to minutes or hours. They suggest that the data have home addresses in the system that may be specified by the users to provide the desired accessibility. Beyond this, the statistics of frequency of use could be used to make blocks of information more accessible and thus make the operation more efficient. Whether this approach has been implemented is not indicated.

Several papers have been concerned with the quantitative effect of the file structure on the expected average access time to obtain records from the file. Lowe (54) has written a very good doctoral dissertation in which he characterizes the average access time in terms of the number of index terms per physical record that can be obtained in an inverted file organization, as opposed to a linked-list structure. He then extends his consideration to the effects of truncating the index terms so that the full index term does not necessarily appear within one physical record. This latter portion of his work has been published as a separate paper (55).

Thompson, et al. (81) have produced a paper closely related to Lowe's, in which they are concerned with the number of logical branches that need to be included at a decision point to minimize search time. Although they deal entirely with logical records, the limitation of the number of branches that can be provided at a given decision point is a function also of the physical record size.

Leimkuhler (51) takes a different approach. He considers the probability of use of the logical records and organizes the file into zones of roughly equivalent probabilities, in order to minimize the access time. He proposes a distribution function for the spread of useful material through a file, and he suggests that empirical information indicates the desirability of a two-level organization for scientific literature. In this organization, 15 to 20% of the most useful documents would be examined first and would provide a success probability of approximately 0.67. The search time is apparently considered to be that required for reading and reviewing the documents retrieved rather than the time for machine retrieval of the document references. The additional effort that would be required to retrieve all document references at the same time is not considered.

Although a number of papers relevant to file structure have been published during the year, the most needed paper has not appeared. That paper will provide a unifying quantitative model of file structure that will cover the range from inverted file to linked-list structures and relate the structures to space, search, and update requirements.

File Search

As was indicated earlier, a measure of association is necessary both for determining the logical grouping of the records and for matching those records with requests. In general, the match criterion specified has been an exact match of the logical combination of the specified values of the fields and subfields of a record. Brandhorst (14) has suggested a method of using term weights to bypass the use of Boolean logic. The determination of proper relative term weights appears fairly straightforward and the programming for such an approach reasonably simple. First, the proper Boolean specification is set up. Then arbitrary weight values are assigned to search terms, and the minimum total weight any document must achieve in order to become a "hit" is specified. The following are two examples (the plus denotes union and is read as "or," while the dot denotes intersection and is read as "and"):

$(A + B) + (C \cdot D)$ A = 2, B = 2, C = 1, D = 1, WEIGHT Minimum = 2
$(A + B) \cdot (C \cdot D)$ A = 1, B = 1, C = 2, D = 2, WEIGHT Minimum = 5

Brown (15) reports on the use of a retrieval system with approximately 60,000 compounds in operation at Eli Lilly. While there are some logical combinations of search terms that cannot adequately be reflected in the weighting technique, such as a union of intersections—$(A \cdot B) + (C \cdot D)$, the technique may still be useful for operating retrieval systems.

The translation of the request from the user's language into the structure required for the search system is a continuing problem of search specification. Much work has been put into developing systems to handle the natural language of the system user. Since the general problem has been so intractable, the current emphasis is upon the use of limited subsets of natural language.

Kellogg (43, 44) and Tonge (83) are both concerned with limited processes for translating from the user's language to the system's internal search control. Both are concerned with the search of relational data files and both have arrived at an approach by which the user is provided with the translation of his request for his approval or modification prior to search. Kellogg's system, CONVERSE, is being constructed at System Development Corporation (SDC) under the time-sharing system implemented on the IBM Q-32. It appears that the portion of the CONVERSE program concerned with the translation of the question to the search control is in operation. It is not clear whether searches are being driven by the control information obtained. Tonge's work appears to be in approximately the same stage of development. It will be interesting to follow both approaches as they develop.

A problem usually encountered in systems using a subset of natural language is the tendency of the user to put in additional words that are unnecessary and not defined in the system. It is desirable to have the facility within the system to ignore such words. Easy English, under development at the Moore School and reported by Cautin, et al. (19), removes noise words and questions the user when it encounters words that it cannot recognize. Such facility will be most important for the infrequent users of the console system.

Closely related to the specification of the search is the strategy of carrying out the search. Where probabilistic or associative indexing is utilized, the opportunity exists for modifying the search request on the basis of the results obtained. Another search of the file can then be made with the new request. One of the most interesting programs in this area is that of Salton's group (71) in which the measure of association is the cosine of the angle between the vectors represented by the index terms assigned to the request and the file record. The subsequent automatic adjustment of the retrieval request vector is achieved by modifying the vector with another vector chosen to maximize the difference between the retrieved documents that the user felt to be relevant and those that he deemed irrelevant. The results to date indicate that it is possible to develop procedures for controlled search in a file with associative indexing. Thus, one does not have to obtain ever-larger amounts of material in an ever-larger sphere around the original request. As file sizes increase and as associative techniques are used more, greater attention will be given to search strategies and their relation to the objectives of the file system.

List-Oriented Programming Languages

As file systems deal with the variability in nonnumeric records, more complex data structures are required. As complex structures are incorporated into general-purpose systems, list-processing techniques will have to be incorporated into the system in order to provide satisfactory manipulation of these records.

Foster (30) has written a very good monograph on list processing. He discusses the various aspects of these techniques: the data representation, the operators available in the system, and special features such as "garbage collection." This brief monograph is particularly useful because of his use of ALGOL as a single language in which to define the list-processing operations. He also discusses some of the well-known list-processing languages, such as IPL-V, LISP and COMIT.

List-processing systems have had a significant impact on standard computer languages. Lawson (47) discusses list processing in PL/1. While the facility in PL/1 will certainly be most useful, it is well to note the caution with which Lawson concludes his summary: "The user should be careful to understand the structure and content of all list elements, since he may inadvertently reference a value which has differing characteristics from the based variable declaration." This warning is particularly apt, since the user is developing his own list-processing macros out of the basic PL/1 language. Thus, he must maintain the caution appropriate to any systems programming.

The development of new list-processing languages is continuing. Blackwell (8) provides a brief discussion of a new list-processing language, LOLITA. This language is being implemented in the Culler-Fried system, which uses two keyboards and a display scope connected to a Bunker-Ramo 340 computer. Many of the list-processing operations are controlled by the console operator through a set of function keys. Blackwell discusses the various function keys available to the operator. With these function keys he can define, load, store, or concatenate list symbols. The major difference between LOLITA and other languages, such as LISP or IPL-V, is the availability of these function operators to the man at the console. It appears that the language structure is similar to an elementary CAI language, but with the addition of a number of special-purpose algebraic operators to make the system amenable to computational applications.

The investigation of list-processing techniques is continuing at the Moore School of Electrical Engineering. Carr & Gray (18) report on the current status of this work, in which they are attempting to extend the limits of list processing in several directions. Among the issues they are investigating are: the Growing Machine, a relocatable software system with simple procedures for addition; an "all-pushdown" computer; and a software system to produce software to specification. This project is quite ambitious, and it should produce results of importance to file systems.

FILE SYSTEMS

The previous section reviewed the contributions that have been made to individual aspects of file systems. This section reviews the operating file systems on which there have been reports during the past year.

The section is divided into two parts—Data Management Systems and Special-Purpose Systems. Data management systems are those designed to be general purpose in the sense that they can be used to maintain and search files with many different types of information in different file structures. Special-purpose systems are those developed for specific purposes, even though they may have some generality. The discussion of these systems is divided into two parts, dealing, respectively, with chemical information systems and other systems.

Data Management Systems

Most of the literature of the current year consists of reports on data management systems that have been previously announced. Olle (66, 67) has written two review articles in which he discusses GIS, IDS, INFOL, and TDMS. Though brief, they serve as a good introduction for those who wish to know how these systems relate to each other.

The Time-Shared Data Management System (TDMS) is continuing under development at SDC (91). This system, an outgrowth of the TSS-LUCID system, was discussed by Minker & Sable (61) and by Bleier (9). The latter article provides information on the data structure definition of TDMS. Data structure definition is critical to a data management system, since it determines the amount of structural variation the data can have and implies the corresponding processing effort required. From Bleier's article, the TDMS data structure appears to be quite flexible from the point of view of the definition of the individual logical record. Variably appearing fields and subfields can be maintained and searched with no difficulty. Also, it appears that variable-length fields can be handled. With respect to the logical grouping of records, indexes—referred to as concordances—can be maintained as specified by the user. However, Bleier does not discuss the structure of this concordance or the method of specifying which elements of a logical record will be included in concordances.

Williams & Bartram (93) discuss another aspect of TDMS, that of report generation. The report generation program is called COMPOSE/PRODUCE. The COMPOSE phase of the program is used for the on-line development of the report specification. The PRODUCE phase of the program obtains the information from the report files and produces the report desired. This phase of the process is carried on with little or no interaction with the user. From the paper it appears that the capabilities of the report generator are standard. The significant difference between it and traditional report generators lies in the user interaction in the COMPOSE portion of the program. There is little information about the nature of this interaction beyond the statement that there has been an attempt to use a restricted subset of English as the control language for the console user. In the examples given, this language appears to be very much like COBOL in the English it uses. There is no indication whether the sophisticated user can bypass the verbosity required by such an approach.

Another group of SDC personnel have reported upon the problems of statistical routines for data analysis in an on-line system. Shure, et al. (74) call these routines TRACE, for Time-Shared Routines for Analysis, Classification, and Evaluation. This system is implemented on the Q-32 Time-Sharing System. As the authors indicate, such on-line general analysis programs will be of considerable importance to the researcher who is attempting to perform on-line analysis of very large quantities of information such as that derived from experiments in the behavioral sciences. As a result, it would seem most desirable to incorporate this facility into TDMS.

In last year's Annual Review, Minker & Sable discuss several other data management systems that were in the developmental stage, including DM-1, ADAM, and COLINGO. Dixon & Sable (29) provide a general discussion of DM-1 and its current capabilities. They do not indicate the stage of development of DM-1, but their discussion in the present tense implies that DM-1 exists. In providing an overview of the system, they discuss briefly the languages used, the organization of the indexes needed to provide for record definition and maintenance, the types of program tasks that can be carried out, and the search capability of the system. The structure of the indexing provided is such that records with variable-length fields and nested sub-fields can be specified to the system. Linkage of separate records can also be performed. Thus, for example, the specification of the vendor in an inventory record might be made by the vendor number. This provides a reference to a separate vendor record in which the vendor's name, address, etc., are provided. Programs would then access both records, to retrieve needed information. To indicate the flexibility of the structure, it appears that the Multi-list structure could be duplicated using this facility.

It is interesting to compare the index structure used in DM-1 with the index structure provided in TDMS. On the surface they are very similar, in that each goes through several decoding stages to convert a field name to a code representing the position of that field and then to a record. To control the highly variable data structure, every field in a record is separately indexed so that access to the records in the file can be obtained via any of the logical fields defined.

In DM-1, it seems that there is no way to define certain fields as report fields only. If so, there would be an unnecessary expansion caused by including these fields in the indexes.

Connors (25) provides an initial discussion of some of the experience and results with ADAM (Advanced Data Management), the MITRE Corporation's experimental general information processing system. Char & Foreman (21) provide a final report of the experiments utilizing ADAM. These articles are refreshingly candid, and the system planner should read them carefully before embarking on the development of a data management system.

One of the central problems mentioned is that a generalized system requires a very sophisticated user if some ridiculously excessive machine times are to be avoided. Connors indicates these requirements: "The user must be aware of the implications of how his data is structured; he must understand and control the optimization." He then goes on to give an example of a trivial but actual user-implemented problem—to multiply price by quantity in order to obtain total price. Unfortunately, as a result of the combination of the data organization and the file generation statements he used, the fields were placed into different files and more than one hour of running time was consumed on the IBM 7030 before the procedure was killed. Connors points out that the user can specify a different data structure that would make this search and computation faster, but that he might at the same time be making the procedure more difficult for other uses of the information. Only the user is in the position to estimate what the mix of his processing is going to be, and therefore only he can structure his control of the system. All of this is based upon the assumption that he knows how the system will perform in response to the processes that he specifies, and thus can control the system to make it effective in response to his requirements.

Char & Foreman also encountered the problem of the sophisticated user. Plans were developed for the installation of a remote station at Headquarters, Air Force Logistics Command. This terminal was going to be used so that all committee members, after training, could be making their own queries and interrogations of the system.

> Although a number of attempts were made by most members of the committee at writing their own queries, they were successful at only the simplest queries, and then quite often made mistakes of punctuation or in typing. In nearly all cases, the mission personnel were inclined to turn their more complex queries over to a committee member who was a programmer and had mastered the Fable language and remote equipment to a greater degree . . . some mission personnel, as a matter of principle, did not believe it should be their function to learn a programming language, and others who would try did not have the time and/or patience to master the intricacies of the language and equipment, and became discouraged after repeated failure in attempting to use the English like syntax query language. (p. 52)

When taken out of context these remarks may seem very negative with respect to the capability of the ADAM system. However, the ADAM system is reasonably capable as a data management system. In addition, the experiment was well run, and its findings do indicate a need for continued development of data management systems. There are, however, serious problems in the proper use of such systems, and further controlled experimentation with them is needed to provide guidelines for their development and use.

Another line of activity at MITRE has been the development of AESOP (An Evolutionary System for On-line Processing). Summers & Bennett (78) provide a final report on this prototype interactive system. AESOP is an experimental system implemented on the IBM 7030 (STRETCH) computer and using CRT consoles. Although the article describes AESOP as a data-base-oriented system, it is concerned with the terminal control aspects of the system, display, editing, printing, and programming. The article gives no information about the system's data management capability.

Two new data management systems implemented on IBM 1400-series systems, have been reported, along with additional reporting on two earlier systems. The new systems are C-10 and GIPSY (Generalized Information Processing System). The two older systems are MADAM (Moderately

Advanced Data Management) and CFSS (Combined File Search System).

Steil (77) describes work that was originally intended to implement MITRE's COLINGO (now called C-10) as a new version on the 1410. He gives a very good history of the sequence in development objectives and the changes in direction that took place as a result of their findings. For example, they decided to program the system in a LISP-like interpreter. When they began debugging their programs, they found that the performance was off by several orders of magnitude. As a result, what began as the development of another file management system became an experiment to improve the performance of the LISP-like language. The paper is of interest because the author chooses to discuss some of the things that were done *wrong* as well as those things that were done right.

GIPSY is being used by the International Atomic Energy Agency in Vienna. Del Bigio (27) has developed a program manual for the system, which was developed to process bibliographic information. Because of this original objective, it has certain rigidities in terms of the way that fields are identified. However, it does include the capability for variable-length fields and repeating fields, and it permits specification of the fields that will serve as indexes to the records. While a program manual is not meant for casual reading, any who are interested in implementing an IBM 1400 file management system would benefit from this document.

Freeman & Atherton (32) report on the application of another file management system for the IBM 1400 to the problem of bibliographic records. Specifically, they are concerned with organizing and searching document files utilizing CFSS, previously discussed by Climenson (23). Although the primary interest of the authors is the use of the Universal Decimal Classification for retrieval purposes, they indicate briefly the method by which the file was mapped into CFSS and the characteristics of the system that made it beneficial to use for the experimental system that they set up. CFSS is being reprogrammed for the IBM 360 by Service Bureau Corporation. Its availability should be checked for the next review article.

Franks (31) has provided a nontechnical article on the characteristics of the MADAM system. He indicates that MADAM is now available on the IBM 360/30, but does not indicate whether this is through emulation or reprogramming.

Given the development of on-line data management systems and the proliferation of 1401-based systems, what else could possibly be done in the development of general systems? Vinsonhaler (89, 90) has written a system in FORTRAN IV. His system, BIRS (Basic Indexing and Retrieval System), is used primarily for document control. The system is organized to utilize a 32,000-word core machine and six tape drives. The description given is for the implementation on a CDC 3600 at Michigan State University. This system provides for access to the file by index keys. Such access can be on the basis of providing a printed listing of indexes, or by automated search of the file. BIRS/II has been released for national distribution. In addition to the capabilities of BIRS/I, it will include the ability to handle coordinate and weighted indexing and will incorporate sophisticated relevance-searching systems. Finally, it will include word-root analysis and a synonym dictionary for improved searching capability. While the inherent problems of tape-oriented systems for machine searching have been well documented, there are at the same time many applications in which such systems are extremely useful. In this situation, the BIRS/II system will provide a rapid method for a person to try out this method of information control on his own files.

Special-Purpose Systems

Chemical Information Systems. Several of the organizations that are developing chemical information systems have been mentioned in the earlier description of the representation of chemical compounds. Hoffman (38) discusses the chemical structure storage and search system now operating at Du Pont. Climenson (23) discussed an early version of this system. To provide access to the basic documents of the file, there are four separate files: the compound file, the general term file, the thesaurus file, and the fragment file. Chemical structures are stored in a separate registry file, which is used for topological substructure search. Hoffman gives more recent information on the file characteristics, size, growth rate, the cost of data preparation, and the cost of file search operations. The registry files contain 69,000 compounds, and the total input cost per compound is 88 cents. While the computer cost for searching is, of course, a function of the number of questions per batch, the indications are that for searches of the nonpolymer registry file the cost is $55 (for 55,000 nonpolymers). The incremental cost per search question is estimated to be $12. At the present time, the system is implemented on an IBM 7010 with 1301 disc. During 1967 a major study of file organization was undertaken to lead to the storage of their files on direct access devices in preparation for on-line searching. No report is available yet.

Registration and searching of chemical compounds is performed using conventions that define a connection number and oxidation state for each item. Polymers are described in terms of significant repeating units and end groups. Characters in each registry record indicate the presence or absence of information relating to the compound in the document system files.

Van Meter, et al. (87) describe an experimental chemical information system developed under the Army Chemical Information and Data Systems Organization. The goal of the work is practical: to develop a system to satisfy the needs of the Army for chemical information and data. The input for the system consists of 230,000 chemical compounds from three separate files—the Toxicological Information Center of Edgewood Arsenal (TOXINFO) file, the Chemical-Biological Coordination Center (CBCC) file, and the Chemical Abstracts Service (CAS) file. The contents of the system include the molecular formula, structural formula images, connection table representation of structural formulas, CIDS registry number, nomenclature, literature references, file code, and file registry number. The paper is heavily oriented toward the logical aspects of representing information and the nature of the searches that are possible as a result of the logical information included in the file. The physical aspects of the operation are not included. The authors indicate that these aspects will be discussed in a forthcoming report on the system.

Matthews & Thomson (59) discuss briefly a COBOL-organized system to carry out weighted coordinate term search utilizing an index stored on magnetic tape. A weighted search is based on selecting a group of search terms that express each concept of the inquiry, assigning each a weight indicative of its relative importance, and computing a "score" for answers resulting from combinations of these terms. While it may appear that this search method is similar to that proposed by Brandhorst, that is not the intention. Rather, the logic is a conjunction of the terms, and the weighting indicates the relative importance the user attaches to these individual terms. This system has been used to retrieve patents from the Information for Industry patent file. It is reported that "A typical question with ten terms having an average frequency of posting would require 12 minutes on an IBM 1410 (40K) computer, for which current charges are $15. This time assumes that the inquiry is one of about five inquiries which have been processed together." Such information would be more meaningful if it indicated also the size of the tape file over which the searches are being processed.

Other Systems. Kessler (45) gives an overview of the on-line aspects of project TIP (Technical Information Program) at M.I.T. There are now 60,000 articles, covering 32 of the physics journals, in the system. Half are on disc, and half are on tape ready to go to disc. Access over the system is to citations backed up by two microfilm-based printout facilities for access to the documents. Through Project MAC, the experiment has a wide range of users because access to the IBM 7094-based system is through 150 on-line keyboards.

Anderson, et al. (1) discuss the experimental literature system that is available for reference retrieval experimentation at Lehigh University's

Center for the Information Sciences. The file at the present time contains 2,500 document references in the information sciences. These are being transferred to disc, and on-line search programs are being set up to operate through GE Data Net-15s and Model 33 Teletypes. The access will be to both serial and inverted files, which will be maintained on the GE 225. The search programs will be capable of both exact match and associative searching. The conversation routine for specifying a search uses normal search terms, not search codes. It appears, however, that the searches are placed entirely in terms of conjunction of terms, without "or" logic available.

The Information Systems Laboratory (ISL) of the Moore School of Electrical Engineering, University of Pennsylvania, has been described by Rubinoff (62). One of the interesting assertions made is that computer aid must be provided not only for searching a document file but also for librarian-like assistance that the user may obtain when he is operating in real time with the computer system over his on-line console. In other words, it must also be able to indicate to the user such information as how the file is organized and what the meanings are of index terms that were adopted by indexers at the time of indexing. The planned system will have the following features: The user will have direct console access to the system. He will be able to obtain not only catalog and index data, but descriptions of the system itself. He will be permitted an unrestricted search vocabulary, and it will be the responsibility of the system to interpret search terms, to request clarifications, or to provide meanings of terms upon request.

Much like Project INTREX, the ISL document system is expected to incorporate many different access points, such as language, color of the document, etc. In its first form, this system is implemented on an IBM 7040 computer with an IBM 1301 disc. A DEC PDP-8 serves as an interface between the users and the 7040 system. This laboratory will be an outstanding place within which to do the research that will continue the excellent reputation of the Moore School.

A set of reports by Magnino (56, 57) and Nelson (65) discuss an operational system that uses text processing of input documents and disseminates the information to 2,400 user profiles for IBM and World Trade Corporation. The text processing is based upon the entire amount of text available in machine form. In most cases this is an abstract of 200 to 300 words, plus title and other bibliographic data. It appears that searching is based upon matching the text of the documents with the precise search terms provided by the user. This requires the user to set up all of the alternative terms that he wishes to have used in the search.

From an input of 12,500 documents, more than 500,000 abstracts were sent out during 1966, to approximately 1,400 users. Thus the ratio of notifications per document per user was .030, meaning that the average user received abstracts of 3% of the documents announced during the year. The 1966 data provided show that over 83% of the notifications (on a 70% return of evaluation information) were judged by the users to be relevant to their interests. While this judgment of relevance is extremely different from the relevance determined when a person is making a specific search request, it shows that the system is providing information the user wants to see. It is most encouraging that an operation of this magnitude is providing useful service to its users.

Two organizations that have been mentioned several times are MITRE and Cornell University. Both organizations have made significant contributions to file organization and search, and both have embarked upon significant efforts in text processing in file systems. From the two documents provided on SAFARI (20, 92), the present concerns of investigators are for text analysis and they have not attempted to incorporate search and retrieval capabilities. By contrast, Salton's work at Cornell is well advanced. The SMART system is a proven research vehicle within which the text processing is being incorporated. In the past, excellent work has been provided by the many organizations in the field, and it is likely that as they enter new areas of inquiry, they will similarly make further significant contributions.

CONCLUSION

Because this has been a review of a single year's literature, it has necessarily covered only small portions of many ongoing projects. As a result, it is difficult to isolate a single "significant event" for the year. Nevertheless, it is possible to derive an implied significance for the file system literature of 1967 by projecting what should occur within the next three years.

All programs in information processing education will incorporate at least one course devoted to the creation and maintenance of file systems. Quantitative models of file systems will incorporate characteristics of both structure and use and will be used to predict the following: response time, storage requirement, processing effort, and transaction capacity. Finally, operational file systems will commonly utilize statistical analysis, such as that of the co-occurrence of indexing terms, to improve their retrieval performance.

In three years, the principal research concern will be user interaction with on-line graphic display file systems. As a result of its dynamic aspect, this research will necessitate further work on the issues of structure, search strategy, quantitative analysis, and, of course, education. Overall, our understanding of file systems is reasonably good. The work, in fact, is more extensive and better than this reviewer anticipated. There are now a number of strong individuals and groups concerned with file systems. From this, we can anticipate that knowledge and application will be extended rapidly.

REFERENCES

(1) ANDERSON, RONALD R.; AMICO, ANTHONY F.; GREEN, JAMES S. Experimental retrieval systems studies, Report no. 2: Systems manual for experimental literature collection and reference retrieval system. Center for the Information Sciences, Lehigh University, Bethlehem, Pa., 15 April 1967, 59 p. (AFOSR-724-65) (AD-653 279)

(2) ANDERSON, RONALD R.; KASARDA, ANDREW J.; REED, DAVID M. Experimental retrieval systems studies. Report no. 3. Center for the Information Sciences, Lehigh University, Bethlehem, Pa., 15 April 1967, 88 p. (NSF-GE-2569) (AD-653 280)

(3) ARMITAGE, JANET E.; CROWE, J. E.; EVANS, P. N.; LYNCH, M. F.; McGUIRK, J. A. Documentation of chemical reactions by computer analysis of structural changes. Journal of Chemical Documentation, 7 (November 1967) 209–215.

(4) ARMITAGE, JANET E.; LYNCH, MICHAEL F. Articulation in the generation of subject indexes by computer. Journal of Chemical Documentation, 7 (August 1967) 170–178. Presented at the 153rd Meeting of the American Chemical Society, Division of Chemical Literature, Miami, Fla., 9–14 April 1967.

(5) BARRON, D. W.; FRASER, A. G.; HARTLEY, D. F.; LANDY, B.; NEEDHAM, R. M. File handling at Cambridge University. In: AFIPS Conference Proceedings, vol. 30; 1967 Spring Joint Computer Conference, Atlantic City, N.J., 18–20 April. Thompson, Washington, D.C., 1967, p. 163–167.

(6) BEMER, R. W. Do it by the numbers—digital shorthand. Communications of the ACM, 3 (October 1960) 530–536.

(7) BENNER, FRANK H. On designing generalized file records for management information systems. In: AFIPS Conference Proceedings, vol. 31: 1967 Fall Joint Computer Conference, Anaheim, Calif., 14–16 November. Thompson, Washington, D.C., 1967, p. 291–303.

(8) BLACKWELL, FREDERICK W. An on-line symbol manipulation system. In: Proceedings of 22nd National Conference, Association for Computing Machinery. Thompson, Washington, D.C., 1967, p. 203–209.

(9) BLEIER, ROBERT E. Treating hierarchical data structures in the SDC Time-Shared Data Management System (TDMS). System Development Corp., Santa Monica, Calif., 29 August 1967, 23 p. (SP-2750) Also published in: Proceedings of 22nd National Conference, Association for Computing Machinery. Thompson, Washington, D.C., 1967, p. 41–49.

(10) BOBKA, MARILYN E.; SUBRAMANIAM, J. B. A computer oriented scheme for coding chemicals in the field of biomedicine. Center for Documentation and Communication Research, School of Library Science, Case Western Reserve University, Cleveland, Ohio, July 1967, 22 p. (Comparative Systems Lab. Technical report no. 11)

(11) BOBROW, DANIEL G.; MURPHY, DANIEL L. The structure of a LISP system using two-level storage. Bolt Beranek and Newman Inc., Cambridge, Mass., 4 November 1966, 26 p. (Report no. Scientific-6) (AFCRL 66-774) (AD-647 601) Also published in: Communications of the ACM, 10 (March 1967) 155–159.

(12) BOURNE, CHARLES P.; FORD, DONALD F. A study of methods for systematically abbreviating English words and names. Journal of the Association for Computing Machinery, 8 (October 1961) 538–552.

(13) BOWMAN, CARLOS M.; LANDEE, FRANC A.; RESLOCK, MARY H. A

chemically oriented information storage and retrieval system. I: Storage and verification of structural information. Journal of Chemical Documentation, 7 (February 1967) 43–47.

(14) **BRANDHORST, W. T.** Simulation of Boolean logic constraints through the use of term weights. American Documentation, 17 (July 1966) 145–146.

(15) **BROWN, WILLIAM F.** A matrix information storage and retrieval system utilizing an IBM-360, Model-30 Computer. In: Proceedings of the 30th Annual Meeting of the American Documentation Institute, New York, October 1967. Thompson, Washington, D.C., 1967, p. 36–40.

(16) **BRYANT, EDWARD C.; SEARLS, DONALD T.; SHUMWAY, ROBERT H.; WEINMAN, DAVID G.** Associative adjustments to reduce errors in document screening. Westat Research, Inc., Bethesda, Md., 31 March 1967, 78 p. (Final report no. 66–301) (AFOSR 67-0980) (AD-651 630)

(17) **CALESS, T. W.; KIRK, D. B.** An application of UDC to machine searching. Journal of Documentation, 23 (September 1967) 208–215.

(18) **CARR, J. W., III; GRAY, H. J.;** et al. List processing research techniques. Third quarterly report, December 1966–April 1967. Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, September 1967. 120 p. (Moore School Report 68–03) (Technical report ECOM 02377–3)

(19) **CAUTIN, HARVEY; LOWE, THOMAS C.; RAPP, FREDERICKA; RUBINOFF, MORRIS.** An experimental on-line information retrieval system. University of Pennsylvania, Moore School of Electrical Engineering, Philadelphia, April 1967, 107 p.; 1967, 20 p.

(20) **CHAPIN, P. G.; GROSS, L. N.; NORTON, L. M.; BELLER, R. J.; BROWNE, C. T.** SAFARI, an on-line text-processing system user's manual. MITRE Corp., Bedford, Mass. March 1967, 34 p. (Information System Language Studies no. 15) (MTP-60) (MITRE Project 1108)

(21) **CHAR, BEVERLY F.; FOREMAN, ALLING C.** Joint AFLC/ESD/MITRE Advanced Data Management (ADAM) Equipment. MITRE Corp., Bedford, Mass., February 1967, 124 p. (Final report no. MTR-285) (ESD TR-66-330) (AD-648 226)

(22) **CHIEN, R. T.; PREPARATA, F. P.** Topological structures of information retrieval systems. University of Illinois, Coordinated Science Lab., Urbana, October 1966, 19 p. (Report no. R-325) (AD-642 501)

(23) **CLIMENSON, W. DOUGLAS.** File organization and search techniques. In: Annual review of information science and technology. Carlos A. Cuadra, ed. Interscience, New York, 1966, vol. 1, p. 107–135.

(24) **COHEN, JACQUES.** A use of fast and slow memories in list-processing languages. Communications of the ACM, 10 (February 1967) 82–86.

(25) **CONNORS, THOMAS L.** Software concerns in advanced information systems. In: Walker, Donald E., ed. Information system science and technology. Papers prepared for the Third Congress. Thompson, Washington, D.C., 1967, p. 395–398.

(26) **CORNELL UNIVERSITY. DEPARTMENT OF COMPUTER SCIENCE.** Information storage and retrieval. Scientific report no. ISR-12 to the National Science Foundation. Reports on evaluation, clustering, and feedback. Gerard Salton, Project Director. Ithaca, N.Y., June 1967, 1 vol. (various pagings)

(27) **Del BIGIO, G.** GIPSY: Generalized Information Processing System. Program manual. Internal publication of International Atomic Energy Agency, Vienna, Austria, 16 May 1967, 92 p.

(28) **DeMAINE, P. A. D.; KLOSS, K.; MARRON, B. A.** The SOLID system. II: Numeric compression. III: Alphanumeric compression. National Bureau of Standards, Washington, D.C., 15 August 1967. (Technical note 413)

(29) **DIXON, P. J.; SABLE, J.** DM-1: A generalized data management system. In: AFIPS Conference Proceedings, vol. 30; 1967 Spring Joint Computer Conference, Atlantic City, N.J. 18–20 April. Thompson, Washington, D.C., 1967, p. 185–198.

(30) **FOSTER, J. M.** List processing. American Elsevier, London and New York, 1967, 54 p. (Macdonald computer monographs)

(31) FRANKS, E. W. The MADAM System: data management with a small computer. System Development Corp., Santa Monica, Calif., 8 September 1967, 16 p. (SP-2944) (AD-658 472)

(32) FREEMAN, ROBERT; ATHERTON, PAULINE. File organization and search strategy using the Universal Decimal Classification in mechanized reference retrieval systems. American Institute of Physics, UDC Project, New York, 15 September 1967, 30 p. (Report no. AIP/UDC-5) (PB-176 152)

(33) GABRINI, PHILIPPE J. Automatic introduction of information into a remote-access system: a physics library catalog. University of Pennsylvania, Moore School of Electrical Engineering, Philadelphia, 1 November 1966, 79 p. (Technical Report, no. 67–09) (AD-641 564)

(34) GRAUER, ROBERT T.; MESSIER, MICHEL. An evaluation of Rocchio's clustering algorithm. In: Cornell University. Dept. of Computer Science. Information storage and retrieval. Report no. ISR-12 to NSF. Gerard Salton, Director. Ithaca, N.Y., June 1967, sec.-6 (39 p.)

(35) GRAY, J. C. Compound data structure for computer aided design; a survey. In: Proceedings of the 22nd National Conference, Association for Computing Machinery. Thompson, Washington, D.C., 1967, p. 355–365.

(36) HADDON, B. K.; WAITE, W. M. A compaction procedure for variable-length storage elements. Computer Journal, 10 (August 1967) 162–167.

(37) HAYES, ROBERT M. A theory for file organization. In: Karplus, Walter J., ed. On-line computing: time-shared computer systems. McGraw-Hill, New York, 1967, p. 264–289.

(38) HOFFMAN, WARREN S. An integrated chemical structure storage and search system operating at Du Pont. In: American Chemical Society. Abstracts of papers, 154th Meeting, Chicago, 10–15 September 1967. Washington, D.C., 1967, 16 p. (Sec. G-Div. of Chem. Lit., paper 15)

(39) HYDE, E.; MATTHEWS, F. W.; THOMSON, LUCILLE H.; WISWESSER, W. J. Conversion of Wiswesser Notation to a connectivity matrix for organic compounds. Journal of Chemical Documentation, 7 (November 1967) 200–204.

(40) IDE, ELEANOR; WILLIAMSON, R.; WILLIAMSON, D. The Cornell programs for cluster searching and relevance feedback. In: Cornell University. Dept. of Computer Science. Information storage and retrieval. Report no. ISR-12 to NSF. Gerard Salton, Director. Ithaca, N.Y., June 1967, sec.-4 (13 p.)

(41) JONES, PAUL E.; CURTICE, ROBERT M. A framework for comparing term association measures. American Documentation, 18 (July 1967) 153–161.

(42) JONES, PAUL E.; GIULIANO, VINCENT E.; CURTICE, ROBERT M. Papers on automatic language processing. Arthur D. Little, Inc., Cambridge, Mass., February 1967, 3 vols. (ESD TR-67-202, vol. 1–3) Vol. 1: Selected collection statistics and data analyses. (AD-649 073); Vol. 2: Linear models for associative retrieval. (AD-649 038); Vol. 3: Development of string indexing techniques. (AD-649 039)

(43) KELLOGG, CHARLES H. CONVERSE—a system for the on-line description and retrieval of structured data using natural language. System Development Corp., Santa Monica, Calif., 26 May 1967, 16 p. (SP-2635)

(44) KELLOGG, CHARLES H. On-line translation of natural language questions into artificial language queries. System Development Corp., Santa Monica, Calif., 28 April 1967, 47 p. (SP-2827)

(45) KESSLER, M. M. The "on-line" technical information system at M.I.T.—Project TIP. In: 1967 IEEE International Convention Record. Institute of Electrical and Electronics Engineers, New York, 1967, part 10, p. 40–43.

(46) LAMB, SIDNEY M.; JACOBSEN, WILLIAM H., JR. A high speed large capacity dictionary system. Mechanical Translation (MT), 6 (November 1961) 76-107.

(47) LAWSON, HAROLD W., JR. PL/I list processing. Communications of the ACM, 10 (June 1967) 358–367.

(48) LEFKOVITZ, DAVID. A chemical notation and code for computer manipulation. Journal of Chemical Documentation, 7 (November 1967) 186–192.

(49) LEFKOVITZ, DAVID. The impact of third generation ADP equipment on alternative chemical structure information systems. Presented at the 153rd Meeting of the American Chemical Society, Division of Chemical Literature, Miami, Fla., 9–14 April 1967, 31 p.

(50) LEFKOVITZ, DAVID. Use of a nonunique notation in a large-scale chemical information system. Journal of Chemical Documentation, 7 (November 1967) 192–200.

(51) LEIMKUHLER, FERDINAND F. A literature search model. [Abstract] Bulletin of the Operations Research Society of America, 15, supplement 1 (Spring 1967) p. B56–57. (PB 174 390)

(52) LEVIEN, R. E.; MARON, M. E. A computer system for inference execution and data retrieval. Communications of the ACM, 10 (November 1967) 715–721.

(53) LONG, JOHN M.; BARNARD, HOWARD J.; LEVY, GERTRUDE C. Dictionary buildup and stability of word frequency in a specialized medical area. American Documentation, 18 (January 1967) 21–25.

(54) LOWE, THOMAS C. Design principles for an on-line information retrieval system. University of Pennsylvania, Moore School of Electrical Engineering, Philadelphia, December 1966, 136 p. (Technical Report no. 67–14) (AFOSR 67–0423) (AD-647 196)

(55) LOWE, THOMAS L. Direct-access memory retrieval using truncated record names. Software Age, 1 (September 1967) 28–33.

(56) MAGNINO, JOSEPH J., JR. IBM's unique but operational international industrial textual documentation system—ITIRC. Presented at 33rd Conference of FID and International Congress on Documentation, Tokyo, 12–22 September 1967. (Preprint, 9 p. and appendix)

(57) MAGNINO, JOSEPH J., JR. Information technology and management science. Paper presented at the Institute of Management Sciences, 14th International Meeting, Mexico City, August 1967. (Preprint, 8 p. and appendix)

(58) MARRON, B. A.; DeMAINE, P. A. D. Automatic data compression. Communications of the ACM, 10 (November 1967) 711–714.

(59) MATTHEWS, F. W.; THOMSON, L. Weighted term search: a computer program for an inverted coordinate index on magnetic tape. Journal of Chemical Documentation, 7 (February 1967) 49–56.

(60) MEADOW, CHARLES T. The analysis of information systems; a programmer's introduction to information retrieval. Wiley, New York, 1967, 301 p. (Information Sciences Series)

(61) MINKER, JACK; SABLE, JEROME. File organization and data management. In: Annual review of information science and technology. Carlos A. Cuadra, ed. Interscience, New York, 1967, vol. 2, p. 123–160.

(62) MOORE SCHOOL OF ELECTRICAL ENGINEERING. The Moore School Information Systems Laboratory. Morris Rubinoff (principal investigator). University of Pennsylvania, Moore School of Electrical Engineering, Philadelphia, May 1967, 28 p. (AFOSR 67-1952) (AD 657 809)

(63) MORENOFF, EDWARD; McLEAN, JOHN B. Application of level changing to a multilevel storage organization. Communications of the ACM, 10 (March 1967) 149–154.

(64) MORENOFF, EDWARD; McLEAN, JOHN B. A code for non-numeric information processing applications in online systems. Communications of the ACM, 10 (January 1967) 19–22.

(65) NELSON, PAUL J. User profiling for normal text retrieval. In: Proceedings of the 30th Annual Meeting of the American Documentation Institute, New York, October 1967. Thompson, Washington, D.C., 1967, p. 288–295.

(66) OLLE, T. WILLIAM. Generalized systems for storing structured variable length data and retrieving information. Presented at IFIP/FID Conference on Mechanized Information Storage and Retrieval, Rome, Italy, 14–17 June 1967. (Preprint, 18 p.)

(67) OLLE, T. WILLIAM. IDS and GIS: Chalk and untasted cheese. Prepared for publication in: Newsletter of the ACM Special Interest Committee in Business Data Processing, 28 September 1967. (Preprint, 7 p.)

(68) POLLOCK, STEPHEN. Measures for the comparison of information retrieval systems, and the normalized sliding ratio. [Abstract] Bulletin of the Operations Research Society of America, 15, supplement 1 (Spring 1967) p. B-57.

(69) PREPARATA, F. P.; CHIEN, R. T. On clustering techniques of citation graphs. University of Illinois, Coordinated Science Lab., May 1967, 25 p. (Report no. R-349) (AD-652 593)

(70) ROSS, DOUGLAS T. The AED free storage package. Communications of the ACM, 10 (August 1967), 481–492.

(71) SALTON, GERARD. The SMART Project—status report and plans. In: Cornell University. Dept. of Computer Science. Information storage and retrieval. Report no. ISR-12 to NSF. Gerard Salton, Director. Ithaca, N.Y., June 1967, sec.-1 (12 p.)

(72) SCHECTER, GEORGE, ed. Information retrieval—a critical view. Based on Third Annual Colloquium on Information Retrieval, 12–13 May 1966, Philadelphia, Pa. Thompson, Washington, D.C., Academic Press, London, 1967, 282 p.

(73) SHUMWAY, R. H. On the expected gain from adjusting matched term retrieval systems. Communications of the ACM, 10 (November 1967) 722–725.

(74) SHURE, GERALD H.; MEEKER, ROBERT J.; MOORE, WILLIAM H., JR. TRACE—Time-shared Routines for Analysis, Classification and Evaluation. In: AFIPS Conference Proceedings, vol. 30; 1967 Spring Joint Computer Conference, Atlantic City, N.J., 18–20 April. Thompson, Washington, D.C., 1967, p. 525–529.

(75) SPARCK JONES, KAREN; JACKSON, DAVID. Current approaches to classification and clump-finding at the Cambridge Language Research Unit. Computer Journal, 10 (May 1967) 29–37.

(76) SPARCK JONES, KAREN; JACKSON, DAVID M. The use of the theory of clumps for information retrieval. Report on the O.S.T.I.-supported project. Cambridge Language Research Unit, Cambridge, England, June 1967, 1 vol. (various pagings) (M.L. 200)

(77) STEIL, GILBERT P., JR. File management on a small computer: the C-10 System. In: AFIPS Conference Proceedings, vol. 30; 1967 Spring Joint Computer Conference, Atlantic City, N.J., 18–20 April. Thompson, Washington, D.C., 1967, p. 199–212.

(78) SUMMERS, J. K.; BENNETT, EDWARD M. AESOP—a final report: a prototype on-line interactive information control system. In: Walker, Donald E., ed. Information system science and technology. Papers prepared for the Third Congress. Thompson, Washington, D.C., 1967, p. 69–86.

(79) SWETS, JOHN A. Effectiveness of information retrieval methods. Bolt Beranek and Newman Inc., Cambridge, Mass., 15 June 1967, 47 p. (AFCRL-67-0412)

(80) TATE, F. A. Handling chemical compounds in information systems. In: Annual review of information science and technology. Carlos A. Cuadra, ed. Interscience, New York, 1967, vol. 2, p. 285–309.

(81) THOMPSON, DAVID A.; BENNIGSON, LAWRENCE; WHITMAN, DAVID. Structuring information bases to minimize user search time. In: Proceedings of the 30th Annual Meeting of the American Documentation Institute, New York, October 1967. Thompson, Washington, D.C., 1967, p. 164–168.

(82) THOMSON, LUCILLE H.; HYDE, E.; MATTHEWS, F. W. Organic search and display using a connectivity matrix derived from Wiswesser Notation. Journal of Chemical Documentation, 7 (November 1967) 204–209.

(83) TONGE, FRED M. A simple scheme for formalizing data retrieval requests. RAND Corp., Santa Monica, Calif., May 1967, 31 p. (Report no. RM-5150-PR) (AD-652 201)

(84) UCHIDA, H.; KIKUCHI, T.; HIRAYAMA, K. Mechanized retrieval system for organic compounds—An evaluation of the fragmentation code system. In: 33rd Conference of FID and International Congress on Documentation, Tokyo, 12–22 September 1967. Abstracts. [Tokyo], 1967, 13 p.

(85) Van DAM, ANDRIES; EVANS, DAVID. A compact data structure for storing, retrieving and manipulating line drawings. In: AFIPS Conference Proceedings, vol. 30; 1967 Spring Joint Computer Conference, Atlantic City, N.J., 18–20 April. Thompson, Washington, D.C., 1967, p. 601–610.

(86) **VANDER STOUW, G. G.; NAZNITSKY, I.; RUSH, J. E.** Procedures for converting systematic names of organic compounds into atom-bond connection tables. Presented at the 153rd Meeting of the American Chemical Society, Division of Chemical Literature, Miami, Fla., 9–14 April 1967.

(87) **VAN METER, CLARENCE T.; LEFKOVITZ, DAVID; POWERS, RUTH V.** An experimental chemical information and data system. Status report, January–December 1966. University of Pennsylvania, Philadelphia, January 1967, 220 p. (Report no. CIDS-4) (Contract DA-18-035-AMC-288 (A)) (AD-657 575)

(88) **VENNER, FRANK H.** On designing generalized file records for management information systems. In: AFIPS Conference Proceedings, vol. 31: 1967 Fall Joint Computer Conference, Anaheim, Calif., 14–16 November. Thompson, Washington, D.C., 1967, p. 291–303.

(89) **VINSONHALER, JOHN F.** BIRS: a system of general purpose computer programs for information retrieval in the behavioral sciences. American Behavioral Scientist, 10 (February 1967) 12, 21–24.

(90) **VINSONHALER, JOHN F.** BIRS: a system of general purpose computer programs for information retrieval. Learning Systems Institute, College of Education, Michigan State University, 9 February 1967, 19 p. (Papers of the Institute #39, revised)

(91) **VORHAUS, ALFRED H.; WILLS, ROBERT D.** The Time-shared Data Management System: a new approach to data management. System Development Corp., Santa Monica, Calif., 13 February 1967, 11 p. (SP-2747)

(92) **WALKER, DONALD E.** SAFARI, an on-line text-processing system. In: Proceedings of the 30th Annual Meeting of the American Documentation Institute, New York, October 1967. Thompson, Washington, D.C., 1967, p. 144–147.

(93) **WILLIAMS, WILLIAM D.; BARTRAM, PHILIP R.** COMPOSE/PRODUCE: A user-oriented report generator capability within the SDC Time-shared Data Management System. In: AFIPS Conference Proceedings, vol. 30: 1967 Spring Joint Computer Conference, Atlantic City, N.J., 18–20 April. Thompson, Washington, D.C., 1967, p. 635–640.

(94) **ZUNDE, PRANAS; ARMSTRONG, FRANCES T.; STRETCH, TERRANCE T.** Evaluating and improving internal indexes. In: Proceedings of the 30th Annual Meeting of the American Documentation Institute, New York, October 1967. Thompson, Washington, D.C., 1967, p. 86–89.